2016

A GPU-accelerated high-order multiphase computational tool for asteroid fragmentation and pulverization modeling

Ben James Zimmerman
Iowa State University

A GPU-accelerated high-order multiphase computational tool for asteroid fragmentation and pulverization modeling

by

Ben J. Zimmerman

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Aerospace Engineering

Program of Study Committee:

Bong Wie, Co-major Professor

Jonathan Regele, Co-major Professor

Paul Durbin

Wei Hong

Anupam Sharma

Iowa State University

Ames, Iowa

2016

DEDICATION

To Mom and Dad,

who supported me the entire way.

To Joey,

who makes me laugh.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I hold a deep appreciation for those who I have worked with and around throughout my graduate school career. Most importantly, I would like to offer my deepest appreciation to my major professor, Dr. Bong Wie. I am incredibly grateful for having the opportunity to develop something of my very own, and for having a mentor who supported me through it. Thank you for encouraging me to tackle the most difficult of problems and for helping me start my professional career.

Secondly, I would like to thank my co-major professor, Dr. Jonathan Regele, for his guidance and patience. Never before have I seen so much red ink on one of my papers. Thank you so much for taking the time to read my papers, discuss them with me, and improve my scientific writing. I greatly appreciate it.

I would like to thank my committee members for their effort and contributions to this work: Dr. Paul Durbin, Dr. Wei Hong, and Dr. Anupam Sharma. I also would like to thank my colleagues and friends. To Josh and George, whose collaboration and friendship I highly value. From the discussions on astrodynamics and physics to determining whose code worked at the end of the day and whose didn't, the problems we faced and laughs we had were priceless. I am grateful to Bharat, for answering my Linux questions and being my friend in numerical methods. To Dan, who told me to put `-j` in my Makefile. To Jim, who helped me set-up my workstations. To Jackie, who answered all my graduate school questions. To Richard and Fangliang, I don't even know where to start. Thank you so much for the discussions on solid mechanics, all the way from the basics to concepts we scratched our heads about.

ABSTRACT

The impact threat of asteroids or comets, referred to as near-Earth objects (NEOs), is a growing concern to the global community. A NEO collision could have severe consequences, especially in highly populated regions. To combat this threat, several asteroid deflection strategies have been introduced, but computational modeling is needed to investigate the feasibility of such missions. To this end, an asteroid disruption software tool was built to handle the simulation of multiple disruption techniques, namely high-energy explosives and kinetic-energy impactors, and to investigate the feasibility and effectiveness of these approaches. In addition, the software is intended to use graphics processing units (GPUs) as the primary computational resource rather than central processing units (CPUs). While GPUs are quickly becoming an alternative computing platform for numerical simulations, it is not clear which numerical schemes provide the highest computational efficiency for different problem types.

The numerical accuracies and computational work of several numerical methods are compared using GPU computing implementation. The Correction Procedure via Reconstruction (CPR), Discontinuous Galerkin (DG), Nodal Discontinuous Galerkin (NDG), Spectral Difference (SD), and Finite Volume (FV) methods are investigated for smooth and discontinuous problems to determine the most efficient method to apply towards asteroid disruption simulations. The computational time to reach a set error criteria and total time to compute solutions are compared across the methods. It is shown that while FV methods can produce solutions with the lowest computation time for discontinuous problems, they produce larger errors for smooth problems at the same order of accuracy. The SD method illustrates an excellent trade-off in terms of error and total work, computing both smooth and discontinuous problems faster than most other methods while providing low error norms.

From the aforementioned study, the SD method is applied to multifluid modeling for asteroid disruption applications. In order to model the multiple material phases associated with the problem, a Diffused-Interface Method (DIM) approach is integrated into the SD method (SD-DIM). This allows high-order solution reconstructions for problems containing multi-material interactions, where different

equations of states define each material. In addition, a damage model is developed to simulate asteroid fracturing based on material phase changes. This results in a novel GPU-based high-order SD-DIM computational tool to explore the complex problem of asteroid fragmentation and pulverization. Several asteroid disruption simulations are completed, including kinetic-energy impactors, multi-kinetic energy impactor systems, and nuclear options. Results illustrate the benefits of using a multi-kinetic energy system when compared to a single impactor system for non-nuclear options. The effectiveness of nuclear options is also observed, where complete target destruction is shown.

CHAPTER 1. INTRODUCTION

The numerical simulation of complex physical phenomena demands efficient and accurate numerical methods coupled with high performance computing to hasten solution generation. Typically, this is completed utilizing well known numerical methods (Finite Volume (FV) or Finite Element (FE)) implemented with the intent of utilizing central processing unit (CPU) servers. Rather than following this common practice, this thesis investigates graphics processing units (GPUs) with high-order numerical methods to simulate asteroid disruption techniques. The intent is to discover the most efficient method, in terms of computing speeds and overall solution accuracy, and use the method to simulate asteroid disruption problems.

1.1 Numerical Simulation

The desired simulations of asteroid disruption cases demand large computational power, since a single simulation will involve millions of degrees of freedom (DOFs) per equation. In this thesis, DOFs are defined as the total number of points in the computational domain. In order to choose a numerical method for implementation, several factors must be taken into account, including solution accuracy, implementation efficiency, and maximum allowable time-step. Several high-order methods are compared against the FV method to determine the most efficient method. In this thesis, a high-order method indicates a solution reconstruction of 3^{rd} order and higher [1]. The FV method, while capable of achieving high-order reconstruction, becomes costly in terms of memory access, especially for unstructured grids [2]. The solution reconstruction requires information from neighboring elements, and as the order of accuracy is increased, the number of elements required for communication also increases. In contrast, high-order methods only require information at element neighbors. This compact nature is appealing to parallel processing, especially GPU computing.

1.1.1 GPU computing application

Various researchers have explored GPU Compute Unified Device Architecture (CUDA) developed by NVIDIA [3] with different numerical methods. Implementation of the FV method for GPUs has been investigated by Castro *et al.* [4] for the shallow water equations and Obenschain [5] for unstructured meshes. The parallelism of FV per element is limited, as solutions are reconstructed along element edges before the volume integration step. In contrast, high-order methods have multiple solution states within each element, stored at solution points, which increases parallelism per element. The most developed high-order methods to date include Discontinuous Galerkin (DG), Nodal Discontinuous Galerkin (NDG), Correction Procedure via Reconstruction (CPR), and Spectral Difference (SD).

The DG method [6, 7, 8, 9, 10, 11, 12, 13, 14] was the first high-order method introduced to hyperbolic equations, and the leader of high-order methods in compressible flow simulations in aerospace problems. There are multiple approaches to the DG method, depending on how the integration points are chosen. This thesis uses Gauss-Legendre points for DG implementation, which demands computations of surface and volume integrals at each step. This allows for improved accuracy at a cost of increased computational work per step. A more efficient implementation of DG was completed by Hesthaven and Warburton [15], which moved the integration points to element edges (NDG). For an in depth discussion of the implementation of NDG to GPUs, the reader is directed to the paper by Klöckner *et al.* [16]. The CPR method was developed to improve efficiency of other high-order methods [17, 18, 19, 20], which includes the DG method. The CPR approach allows the equations to be solved in differential form, removing the added surface and volume integration computations present in DG. While this increases the computing speed, the method is not as accurate as the DG approach [1]. CPRs application to GPUs was completed by Hoffmann and Zimmerman [21, 22], where significant speed-ups are observed. The SD method is a finite difference-like formulation [23, 24, 25], which uses two sets of points within each element to compute derivatives and update solution states. The SD methods application to GPUs was completed by Zimmerman [26] for a three-dimensional system.

The aforementioned references layout efficient algorithms and implementation techniques for the numerical methods discussed, and compare the speeds from GPU to CPU implementations, where significant speed-up results are shown. While there has been a comparative study done by Yu *et al.* [27] on

two-dimensional high-order methods using a CPU platform, there has been no performance assessment of the different methods using a GPU platform. Furthermore, there has been no performance comparison between high-order methods to FV methods on GPUs. This thesis performs a fair comparison of two-dimensional numerical methods and evaluates the relative performance between them in terms of total computing speed and accuracy with GPUs, following the work done by Zimmerman, Regele, and Wie [28]. To this end, the FV, CPR, DG, NDG, and SD methods are all implemented using GPU CUDA, in similar manners from the references discussed above. The comparison is for the two-dimensional Euler system, for both smooth and discontinuous problems. Each method is compared at the same order of accuracy and same number of degrees of freedom, with the maximum allowable time-step for a given mesh. The time-step plays an important factor when considering the computational work to reach a specified final time, since high-order methods are time-step restricted, and this restriction increases with the order of accuracy of the scheme [29, 30].

1.1.2 Shock capturing

The simulations of discontinuities require appropriate numerical methods, called shock capturing. Typical approaches for shock capturing include two methods, limiting and artificial viscosity [9, 31, 32]. The concept of applying artificial viscosity to suppress oscillations near discontinuities originated from Von Neumann and Richtmyer, and recently this concept has been extended into several high-order methods [33, 34, 35, 36]. However, artificial viscosity requires adding a term into the partial differential equation and selecting an appropriate value for this viscosity. Additionally, the added dissipative model involves solving a second order derivative, which will increase the computational work per time step. Rather than exploring viscosity methods, this thesis implements slope limiting methods. Slope limiting approaches are much simpler, but once applied, the order of the solution is reduced, and high-order reconstructions become obsolete. Hence, a discontinuity detector is desired to locate the solution discontinuities and apply slope limiting only in these regions.

For the FV method, the Monotonic Upstream-Centered Scheme for Conservation Laws (MUSCL) scheme [37, 38, 39] provides a robust and simple implementation to resolve discontinuities. Solution information from neighboring elements provides the information to reconstruct the slopes and control jumps. For high-order methods, multiple approaches using slope limiters have been developed [40,

41, 9]. The slope limiter developed by Cockburn and Shu [9] provides a methodology to limit the order of the solution only near solution discontinuities. This allows a high-order reconstruction in smooth solution regions while resolving discontinuous solutions. Each high-order method implemented will utilize the slope limiter scheme of Cockburn and Shu, while the FV methods will use MUSCL reconstruction.

1.2 Asteroid Disruption Options

The impact threat of near-Earth objects (NEOs) is a concern to the global community, as demonstrated by the Chelyabinsk event (caused by a 17 m meteorite) in Russia on February 15, 2013 and a near miss by asteroid 2012 DA_{14} (~30 m diameter) on the same day. The Chelyabinsk event released roughly 440 kilotons of energy into the upper atmosphere [42] causing damage to over 7,000 structures. These air burst scenarios are the highest threat, as most NEOs lack the material composition or size to survive entry into the Earth's atmosphere [43]. NEOs within this category are of interest in this thesis.

Since thousands of NEOs cross the Earth's orbit [44], a significant interest is growing in the area of asteroid deflection. One of the most common asteroid defense concepts, and regarded as the best option in planetary defense literature, is to change the trajectory of the target NEO by use of an impulsive force [45]. The impulsive force can be delivered by either a kinetic impactor or a standoff nuclear explosion. However, due to the small velocity change induced on the NEO, at least a decade of warning is needed in advance of the impact. In the event of short warning scenarios (less than 5 years) this typical deflection concept would be ineffective. In the short-warning-time frame, significant damage must be applied to the NEO such that it is completely destroyed or the resulting fragments are small enough to disintegrate upon entering the Earth's atmosphere. However, non-ideal fragmentation of the target can occur, where the fragments are of sufficient size and remain on an Earth impacting trajectory. This scenario may lead to increased damage to the Earth [45, 46, 47, 48], implying the need of numerical simulations to further examine short-warning-time disruption missions.

The need for numerical simulations of such problems is driven by the non-ideal fragmentation possibility and modeling uncertainties, such as asteroid characterization. Asteroid target disruption simulations have previously been carried out by Dr. David Dearborn at Lawrence Livermore National

Laboratory. The work was verified and extended by Kaplinger at the Asteroid Deflection Research Center at Iowa State University [49], and pursued further by Premaratne [50]. All previous work used the Smoothed Particle Hydrodynamics (SPH) method for simulation purposes. This current work seeks to further improve the numerical aspects by employing high-order methods. The SPH method, while robust and easy to understand, suffers from neighbor searching and limited orders of accuracy [51, 52, 53]. While data structures can be employed to assist in locating neighboring points, poor resolution of certain processes, such as dynamical instabilities and mixing, is an issue with SPH [52]. The use of grid-based methods can improve on this aspect, and high-order methods can further increase resolution and capture small structures which are normally dissipated by low-order methods. The application of high-order methods to asteroid disruption problems with GPU computing has been investigated first by Zimmerman [54, 55, 56, 57, 58, 59, 60, 61] and is covered in this thesis.

1.2.1 Kinetic-energy impactors

When dealing with NEO targets of relatively small diameters (< 150 m) non-nuclear options, such as kinetic-energy impactors (KEIs), may be feasible for target disruption. The KEI must yield enough energy such that the target NEO is either pulverized or fragmented. These resultant fragments must be small enough to disintegrate upon entry to Earth's atmosphere or have large dispersion speeds in excess of escape velocity to miss the Earth entirely.

This thesis investigates two different impactor systems, a single heavy impactor and a multi-bodied impactor system, designed to impact at hypervelocity (the impactor speed is faster than the speed of sound in the target). The two impactor systems are the Single Kinetic-Energy Impactor Vehicle (SKIV) and the Multiple Kinetic-Energy Impactor Vehicle (MKIV). The SKIV is a single heavy aluminum impactor which will transfer a significant amount of kinetic energy to the asteroid. Due to the immense impact energy the resultant asteroid fragments will have high dispersion speeds. The MKIV system was first proposed by Wie [61] and investigated further by Lyzhoft and Wie [62], which stemmed from work by Wood *et al.* [63]. The concept is to intercept an asteroid target with an array of kinetic impactors, striking the target in multiple locations over the surface. Zimmerman and Wie [57, 58, 60] were the first to simulate the MKIV system against asteroid targets. The distribution of damage across the surface from the MKIV may be more effective at target disruption when compared to the SKIV.

To ensure a fair comparison between the two systems, the total impacting mass is held constant (equivalent kinetic energy). If the SKIV is 5000 kg, then the MKIV may be built of five separate impactors of 1000 kg each. This thesis compares the velocity dispersal speed of asteroid particles of both systems. The final results which yield larger dispersal speeds will be deemed the more effective approach.

1.2.2 Nuclear options

Nuclear explosive devices (NEDs) are the most mass-efficient means for storing energy. This makes nuclear options more effective than non-nuclear options for larger bodies with a short mission lead time [45, 46, 47, 64, 65]. The specific energy transferred from NEDs are greater than the gravitational binding energy of common NEOs, which yields long-term dispersion of fragments along the orbital trajectory [64]. One such nuclear option is a standoff explosion against the NEO, ablating and blowing off a thin layer of the targets surface [46, 47, 66]. However, since the NED is surrounded by empty space, a significant amount of energy is wasted. Improved energy coupling from the NED is observed if the device is buried in the target. The benefit of subsurface explosions is covered in a National Research Council (NRC) report, where depending on the amount of energy emitted by the NED along with is buried depth, it may be up to 20 times more efficient at coupling energy to the system when compared to a surface explosion [42]. Numerical simulations have verified coupling factors at specific depths [58]. While this can maximize the damage from a nuclear device, it is not physically feasible to bury an NED inside an asteroid body. Hence, a concept was developed to mimic a subsurface NED explosion.

The Hypervelocity Asteroid Intercept Vehicle (HAIV) concept was developed as part of a NASA Innovative Advanced Concepts (NIAC) study [67, 68, 69, 70]. The HAIV concept blends a hypervelocity impactor and a NED together. The kinetic impactor first strikes the target to generate a crater. Then, a follower vehicle carrying the NED enters the crater and detonates the NED. While the 20 times efficiency quoted from the NRC report is an ideal case, the HAIV concept presents a non-ideal situation where the explosive is not entirely buried. The crater is partially open to the outside space, where energy emitted from the NED is able to escape, lowering the efficiency of energy coupling [71]. The HAIV concept is simulated and compared against the KEIs in this thesis.

1.3 Fluid Model Assumption

An interesting physical process occurs when objects impact at hypervelocity or when a high-pressure shock front from an NED contacts the surface. The impact energy is so high that the material's strength is very small compared to the stresses imposed, and the material starts to behave more like a fluid than a solid [72]. The following argument is taken from *High Velocity Impact Phenomena* [73]. Consider a small element in the target, which has a net force acting on it. This net force is contributed by the normal and shear stress gradients, which can be estimated by comparing the normal and shear stresses. Consider the normal stress component, whose order is the pressure generated at the shock, following the order of the kinetic energy. For the impact of a solid at speeds of several kilometers per second, the generated pressures are on the order of megabars. These pressure exceed the strengths of materials by several factors of ten, which creates a regime where it is possible to neglect the materials strength effects and treat the solid as a inviscid, compressible fluid. A similar argument can be made for nuclear options, generating extremely high pressures relative to the maximum shear stress in the target, or much greater than the targets ultimate strength [74, 75, 76]. The fluid argument has been adapted by several authors using the above argument [77, 78, 79, 80, 81, 82, 83, 84]. It should be noted that this approximation does fail when the pressure becomes sufficiently low [85, 86].

As a second argument, assume a target object is impacted at hypervelocity, or an NED is detonated against the surface. The resulting impact, or high-pressure shock wave, will generate longitudinal waves (P-waves) and shear waves (S-waves). P-waves travel faster than S-waves [87] (nearly twice as fast), hence a point in the target beyond the impact location will experience a P-wave disturbance before the S-wave arrives. For general hypervelocity impact, the P-wave particle displacement is greater than the maximum compressive strain of the target, thus causing compressive failure in the target. The S-wave propagation is hindered by the compressive failure from the P-wave, and has difficulty traveling through the damaged regions. This allows a fluid-type model to be applied (S-waves do not travel in fluids), since all damage arises from the P-wave. With this assumption, the compressible Euler equations can be applied. While the complete theory of hypervelocity impact or high-energy explosive effects on targets involves melting, vaporization, resolidification, condensation, and phase change kinetics [73], the following serves as a good approximation for the problem.

1.3.1 Multiphase modeling

The numerical simulation of impact phenomenon requires a model for distinguishing multiple materials. In the presented problems, three different phases must be tracked: Aluminum, granite, and empty space. The impactor is a solid body consisting of aluminum, the target asteroid is assumed to be granite composition, and the outside space is modeled as a low density air (setting zero density in a region for Eulerian methods produces infinity). The nuclear device is modeled as air and initialized as a point source in the domain as specified by Needam [88]. This still requires two distinct phases to model, air and the asteroid target (the NED can be modeled with the properties of air).

The modeling of material interfaces is a challenging problem, where improper treatment of material interfaces can lead to non-physical pressure oscillations, even in first order models [89, 90]. The introduction of high-order methods to these simulations further destabilizes the interface pressure due to the high-order interpolation polynomials, leading to numerical instability. For hyperbolic conservation laws, equations in the conservative form best describe the flow to be modeled, but computing multicomponent fluid dynamics in conservative form gives rise to oscillations and inaccuracies near material interfaces [91, 92, 93]. An important note is that these oscillations are not associated with the oscillations produced by high-order numerical schemes. Abgrall and Karni developed quasi-conservative approaches to correctly resolve the material interfaces without oscillations [89, 90]. While this approach is appealing, the idea of strict conservation is lost to better capture the physics of the problem, which may lead to errors near shock fronts (incorrect shock front speeds).

A non-conservative approach may not be the only way to remove the oscillations at the interface fronts. It was shown by Engquist and Sjogreen [94] that the conservation laws can be augmented using extrapolation techniques and stiff source terms. The issue with the prescribed approach is application to multiple dimensions, as extrapolation directions can be difficult to compute. Another related method is to use the fully conservative form throughout the domain except in the locations of the material fronts. Around the material fronts, the governing equations are changed to the non-conservative equations [95, 96]. However, such hybrid schemes also have issues going to multiple dimensions.

The model adapted in this thesis is based on tracking the material interface and maintaining the conservative form of the governing equations. A brief description of different methods to accomplish

this is covered by Allair, Clerc, and Kokh [97] and is summarized here. There are four main methods to track the material interface. The first method discretizes the interface with points which move at each time-step [98, 99]. The second method reconstructs the interface using a color function which takes on values of 0 or 1 for two distinct fluids. Recovering the interface becomes complicated, and is covered in [100, 101, 102]. The third method is the level set method, where reconstruction of the interface is an interpolation problem [103]. These methods are known to be highly accurate, but suffer from high computational complexity when applied to multidimensional problems. The fourth method is the method adapted for use in this thesis, and is based off a spread interface model [97, 104]. Each fluid is allowed a value similar to a color function, where the transition between the two fluids is computed using a mixture model. The form of the equations is closely related to homogenized two-phase flows based on volume fractions [105, 106, 107]. This approach was first used for simple multicomponent gas flows [92, 108], and was further extended to handle different material interfaces [89, 90, 107, 109, 110]. While this approach is less accurate than the other three methods summarized, it is much simpler to implement and relies on physical modeling of the mixture [97]. The spread interface model is defined as the Diffused-Interface Method (DIM) [97, 104] in this thesis (sometimes referred to as the 4-equation model) and is integrated into a high-order method to resolve the material interfaces.

CHAPTER 2. NUMERICAL METHODS

The governing equations can be expressed in the form of a hyperbolic conservation law, written as the following

$$\frac{\partial \boldsymbol{q}}{\partial t} + \vec{\nabla} \cdot \vec{\boldsymbol{F}}(\boldsymbol{q}) = 0 \tag{2.0.1}$$

where $\boldsymbol{q}$ is the state vector and $\vec{\nabla} \cdot \vec{\boldsymbol{F}}(\boldsymbol{q})$ is the divergence of the inviscid flux vector. The system can be expanded as

$$\vec{\nabla} \cdot \vec{\boldsymbol{F}}(\boldsymbol{q}) = \frac{\partial \boldsymbol{f}(\boldsymbol{q})}{\partial x} + \frac{\partial \boldsymbol{g}(\boldsymbol{q})}{\partial y} \tag{2.0.2}$$

For the two-dimensional Euler equations, $\boldsymbol{q}$ is a vector of the conserved variables, given as

$$\boldsymbol{q} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ e \end{pmatrix} \tag{2.0.3}$$

and $\boldsymbol{f}(\boldsymbol{q})$ and $\boldsymbol{g}(\boldsymbol{q})$ are flux vectors written as follows:

$$\boldsymbol{f}(\boldsymbol{q}) = \begin{pmatrix} \rho u \\ p + \rho u^2 \\ \rho u v \\ u(e + p) \end{pmatrix}, \qquad \boldsymbol{g}(\boldsymbol{q}) = \begin{pmatrix} \rho v \\ \rho u v \\ p + \rho v^2 \\ v(e + p) \end{pmatrix} \tag{2.0.4}$$

In Equations (2.0.3) and (2.0.4), ρ is the density, u is the x-direction velocity, v is the y-direction velocity, e is the total energy per unit volume, and p is the pressure. To close the system, an appropriate equation of state is needed, which is of the form

$$p = f(\rho, e) \tag{2.0.5}$$

For the first part of this thesis, the ideal gas equation of state is used, written as

$$p = (\gamma - 1)(e - \frac{1}{2}\rho(u^2 + v^2)) \tag{2.0.6}$$

The computational domain is discretized with two-dimensional non-overlapping quadrilateral elements, each with volume V_m.

2.1 Numerical Schemes

The various numerical methods implemented are derived and discussed here, including Finite Volume (FV), Correction Procedure via Reconstruction (CPR), Discontinuous Galerkin (DG), Nodal Discontinuous Galerkin (NDG), and Spectral Difference (SD). Each high-order method requires various coefficients to be derived to complete reconstructions, interpolations, and derivatives. Several examples of these coefficients are shown in this chapter. For an in depth discussion, the reader is directed to Appendix A.

2.1.1 FV Formulation

In the FV approach, the solution per element takes on an averaged value. Equation (2.0.1) integrated over the elements volume, V_m, as

$$\int_{V_m} \left[\frac{\partial \boldsymbol{q}}{\partial t} + \vec{\nabla} \cdot \vec{\boldsymbol{F}}(\boldsymbol{q}) \right] dV = 0 \tag{2.1.1}$$

The solution average, denoted by $\bar{\boldsymbol{q}}_m$ is then defined as

$$\bar{\boldsymbol{q}}_m = \frac{1}{V_m} \int_{V_m} \bar{\boldsymbol{q}} dV \tag{2.1.2}$$

The solution update can then be written in the following well known form for two-dimensional quadrilateral elements

$$\frac{\partial \bar{\boldsymbol{q}}_{i,j}}{\partial t} + \frac{1}{\Delta x} \left[\boldsymbol{f}_{i+1/2,j} - \boldsymbol{f}_{i-1/2,j} \right] + \frac{1}{\Delta y} \left[\boldsymbol{g}_{i,j+1/2} - \boldsymbol{g}_{i,j-1/2} \right] = 0. \tag{2.1.3}$$

In the above formulation, i is the index in the x-direction, while j is the index in the y-direction. To obtain the flux at an interface (say $\boldsymbol{f}_{i-1/2,j}$, which is the left interface of the element) left and right solutions need to be reconstructed at the elements edge first. In this thesis, both second and third order reconstructions are considered. For a linear reconstruction at the face $(i-1/2, j)$, a left (interior) and right (exterior) solutions are required, written as

$$\bar{\boldsymbol{q}}^L_{i-1/2,j} = \bar{\boldsymbol{q}}_{i,j} - \frac{1}{2} \left(\bar{\boldsymbol{q}}_{i+1,j} - \bar{\boldsymbol{q}}_{i,j} \right) \tag{2.1.4}$$

$$\bar{\boldsymbol{q}}^R_{i-1/2,j} = \bar{\boldsymbol{q}}_{i-1,j} - \frac{1}{2} \left(\bar{\boldsymbol{q}}_{i,j} - \bar{\boldsymbol{q}}_{i-1,j} \right) \tag{2.1.5}$$

where the L and R superscripts denote the left and right reconstructed solutions. Similarly, a third order reconstruction can be written as

$$\bar{\boldsymbol{q}}^{L}_{i-1/2,j} = -\frac{1}{6}\bar{\boldsymbol{q}}_{i+1,j} + \frac{5}{6}\bar{\boldsymbol{q}}_{i,j} + \frac{1}{3}\bar{\boldsymbol{q}}_{i-1,j} \tag{2.1.6}$$

$$\bar{\boldsymbol{q}}^{R}_{i-1/2,j} = -\frac{1}{6}\bar{\boldsymbol{q}}_{i-2,j} + \frac{5}{6}\bar{\boldsymbol{q}}_{i-1,j} + \frac{1}{3}\bar{\boldsymbol{q}}_{i,j} \tag{2.1.7}$$

The operation needs to be completed for each face in the domain. Once left and right solutions are found at each interface, a Riemann problem is solved to determine the flux value at the interface. The averaged solution is then updated via a time-marching scheme.

2.1.2 CPR Formulation

Here, the CPR method is described. The formulation of the CPR method requires the definition of an arbitrary weighting function w. By multiplying the weighting function to Equation (2.0.1) and integrating over the domain, the following is obtained

$$\int_{V_m} \left[\frac{\partial \boldsymbol{q}}{\partial t} + \vec{\nabla} \cdot \vec{\boldsymbol{F}}(\boldsymbol{q}) \right] w dV = 0 \tag{2.1.8}$$

By applying the Gauss divergence theorem, Equation (2.1.8) is expanded to be

$$\int_{V_m} \frac{\partial \boldsymbol{q}}{\partial t} w dV + \int_{\partial V_m} w \vec{\boldsymbol{F}}(\boldsymbol{q}) \cdot \boldsymbol{n} dS - \int_{V_m} \vec{\nabla} w \cdot \vec{\boldsymbol{F}}(\boldsymbol{q}) dV = 0 \tag{2.1.9}$$

where $\boldsymbol{n}$ is the normal vector at an elements face. Let $\boldsymbol{q}_m$ approximate the solution $\boldsymbol{q}$ within the element V_m. Furthermore, the solution is assumed to belong to the space of polynomials of degree k or less ($\boldsymbol{q}_m \in P^k$). Thus, Equation (2.1.9) must satisfy the following

$$\int_{V_m} \frac{\partial \boldsymbol{q}_m}{\partial t} w dV + \int_{\partial V_m} w \vec{\boldsymbol{F}}(\boldsymbol{q}_m) \cdot \boldsymbol{n} dS - \int_{V_m} \vec{\nabla} w \cdot \vec{\boldsymbol{F}}(\boldsymbol{q}_m) dV = 0 \tag{2.1.10}$$

There is no requirement enforced on element edges at this point. The normal flux is replaced with a common Riemann flux to enforce element coupling

$$\int_{V_m} \frac{\partial \boldsymbol{q}_m}{\partial t} w dV + \int_{\partial V_m} w \vec{\boldsymbol{F}}^{n}_{com}(\boldsymbol{q}_m, \boldsymbol{q}_{m+}) dS - \int_{V_m} \vec{\nabla} w \cdot \vec{\boldsymbol{F}}(\boldsymbol{q}_m) dV = 0 \tag{2.1.11}$$

In Equation (2.1.11), $\boldsymbol{q}_{m+}$ is the solution outside of element m. Next, integration by parts is applied again to the last term in Equation (2.1.11) to yield

$$\int_{V_m} \frac{\partial \boldsymbol{q}_m}{\partial t} w dV + \int_{V_m} w \vec{\nabla} \cdot \vec{\boldsymbol{F}}(\boldsymbol{q}_m) dV + \int_{\partial V_m} w \left[\boldsymbol{F}^{n}_{com} - \boldsymbol{F}^{n}(\boldsymbol{q}_m) \right] dS = 0 \tag{2.1.12}$$

In the CPR formulation, the last term in Equation (2.1.12) is viewed as a penalty term, which can be lifted to a volume integral by introducing a correction polynomial $\boldsymbol{\delta}_m \in P^k$

$$\int_{V_m} w\boldsymbol{\delta}_m dV = \int_{\partial V_m} w\left[\boldsymbol{F}^n_{com} - \boldsymbol{F}^n(\boldsymbol{q}_m)\right] dS \tag{2.1.13}$$

The volume integral formulation of Equation (2.1.12) is obtained

$$\int_{V_m} \left[\frac{\partial \boldsymbol{q}_m}{\partial t} + \vec{\nabla}\cdot\vec{\boldsymbol{F}}(\boldsymbol{q}_m) + \boldsymbol{\delta}_m\right] w dV = 0 \tag{2.1.14}$$

If the conservation law is non-linear, then $\vec{\nabla}\cdot\vec{\boldsymbol{F}}(\boldsymbol{q}_m)$ does not generally fall into the space P^k. To resolve the non-linear situation, the term $\vec{\nabla}\cdot\vec{\boldsymbol{F}}(\boldsymbol{q}_m)$ is projected into P^k. Then, eliminating the weight and volume integral gives the differential formulation

$$\frac{\partial \boldsymbol{q}_m}{\partial t} + \Pi\left[\vec{\nabla}\cdot\vec{\boldsymbol{F}}(\boldsymbol{q}_m)\right] + \boldsymbol{\delta}_m = 0 \tag{2.1.15}$$

where the term $\Pi\left[\vec{\nabla}\cdot\vec{\boldsymbol{F}}(\boldsymbol{q}_m)\right]$ is the projection of the flux divergence. The weighted residual formulation is reduced to a differential one. Each element must store the solution states at a set of points, called solution points. For the CPR method, within an element V_m, a set of Legendre-Lobatto solution points are defined, as shown in Figure 2.1 (b) and (e). At each solution point j, Equation (2.1.15) must be true

$$\frac{\partial \boldsymbol{q}_{m,j}}{\partial t} + \Pi_j\left[\vec{\nabla}\cdot\vec{\boldsymbol{F}}(\boldsymbol{q}_m)\right] + \boldsymbol{\delta}_{m,j} = 0 \tag{2.1.16}$$

Now the calculation of both $\Pi_j\left[\vec{\nabla}\cdot\vec{\boldsymbol{F}}(\boldsymbol{q}_m)\right]$ and $\boldsymbol{\delta}_{m,j}$ must be completed. The inviscid flux divergence follows a chain rule approach, given as

$$\vec{\nabla}\cdot\vec{\boldsymbol{F}}(\boldsymbol{q}_m) = \frac{\partial \boldsymbol{f}(\boldsymbol{q}_{m,j})}{\partial x} + \frac{\partial \boldsymbol{g}(\boldsymbol{q}_{m,j})}{\partial y} \tag{2.1.17}$$

$$= \frac{\partial \boldsymbol{f}(\boldsymbol{q}_{m,j})}{\partial \boldsymbol{q}}\frac{\partial \boldsymbol{q}_{m,j}}{\partial x} + \frac{\partial \boldsymbol{g}(\boldsymbol{q}_{m,j})}{\partial \boldsymbol{q}}\frac{\partial \boldsymbol{q}_{m,j}}{\partial y} \tag{2.1.18}$$

The analytical flux derivative for the Euler system can be computed in the following manner [111]:

$$\frac{\partial \boldsymbol{f}(\boldsymbol{q}_{m,j})}{\partial \boldsymbol{q}} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{\gamma-1}{2}(u^2+v^2) - u^2 & (3-\gamma)u & -(\gamma-1)v & \gamma-1 \\ -uv & v & u & 0 \\ \left[-\frac{\gamma e}{\rho} + (\gamma-1)(u^2+v^2)\right]u & \frac{\gamma e}{\rho} - \frac{(\gamma-1)}{2}(3u^2+v^2) & -(\gamma-1)uv & \gamma u \end{pmatrix} \tag{2.1.19}$$

Table 2.1: Correction coefficients for P^1 - P^4 ($\alpha_{R,j}$) reconstructions.

j	P^1	P^2	P^3	P^4
1	2.0	4.5	8.0	12.0
2	-1.0	-0.75	-0.5938	-0.2612
3	-	1.5	0.9688	0.9375
4	-	-	-2.0	-1.1451
5	-	-	-	0.5

$$\frac{\partial \boldsymbol{g}(\boldsymbol{q}_{m,j})}{\partial \boldsymbol{q}} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ -uv & v & u & 0 \\ \frac{\gamma-1}{2}(u^2+v^2)-v^2 & -(\gamma-1)u & (3-\gamma)v & \gamma-1 \\ \left[-\frac{\gamma e}{\rho}+(\gamma-1)(u^2+v^2)\right]v & -(\gamma-1)uv & \frac{\gamma e}{\rho}-\frac{(\gamma-1)}{2}(u^2+3v^2) & \gamma v \end{pmatrix} \tag{2.1.20}$$

The solution derivatives are computed using a Lagrange polynomial interpolation, written as the following

$$\vec{\nabla}\boldsymbol{q}_{m,j} = \sum_{j=1}^{k+1} \boldsymbol{q}_{m,j}\vec{\nabla} l_j \tag{2.1.21}$$

where l_j are the Lagrange polynomials at the solution points. For $\boldsymbol{\delta}_m$ formulations, the g_{DG} scheme is adapted [17, 19]. This scheme uses Radau polynomials to build the correction polynomial, which results in the following scheme for one-dimensional conservation laws

$$\boldsymbol{\delta}_{m,j} = \alpha_{L,j}\left(\boldsymbol{f}^n_{com} - \boldsymbol{f}^n(\boldsymbol{q})\right)_L + \alpha_{R,j}\left(\boldsymbol{f}^n_{com} - \boldsymbol{f}^n(\boldsymbol{q})\right)_R \tag{2.1.22}$$

where α_L and α_R are correction coefficients for the left and right interfaces respectfully. The only information needed on the edges are the normal flux differences, then the correction polynomial can be built across points on edges. The definition of Legendre-Lobatto solution points brings a sense of efficiency into the method. The solution points occupy edges of elements, thus no interpolation of information to element edges is required, and element coupling becomes straightforward.

With quadrilateral elements, the operations are completed in a one-dimensional manner. Let (i, k) denote the solution points in (x, y) directions respectfully. Then the formulation becomes

$$\begin{aligned}\frac{\partial \boldsymbol{q}_{m:i,k}}{\partial t} &+ \Pi_{i,k}\left[\vec{\nabla}\cdot\vec{\boldsymbol{F}}(\boldsymbol{q}_m)\right] \\ &+ \frac{1}{|V_m|}\Big[\alpha_{R,i}\left(\boldsymbol{f}^n_{com} - \boldsymbol{f}^n(\boldsymbol{q})\right)_{R,i} S_1 + \alpha_{R,k}\left(\boldsymbol{g}^n_{com} - \boldsymbol{g}^n(\boldsymbol{q})\right)_{R,k} S_2 \\ &+ \alpha_{L,i}\left(\boldsymbol{f}^n_{com} - \boldsymbol{f}^n(\boldsymbol{q})\right)_{L,i} S_3 + \alpha_{L,k}\left(\boldsymbol{g}^n_{com} - \boldsymbol{g}^n(\boldsymbol{q})\right)_{L,k} S_4\Big] = 0\end{aligned} \tag{2.1.23}$$

where $|V_m|$ is the volume of element m, S_f is the area of face f, and α are the correction coefficients shown in Table 2.1. The coefficients are symmetric so only one set of coefficients are shown [17].

2.1.3 DG Formulation

The DG method's formulation is more straightforward than the CPR method. Again, a weighting function w multiplies the conservation law, Equation (2.0.1), and is integrated over the domain as

$$\int_{V_m}\left[\frac{\partial \boldsymbol{q}}{\partial t} + \vec{\nabla}\cdot\vec{\boldsymbol{F}}(\boldsymbol{q})\right] w dV = 0 \tag{2.1.24}$$

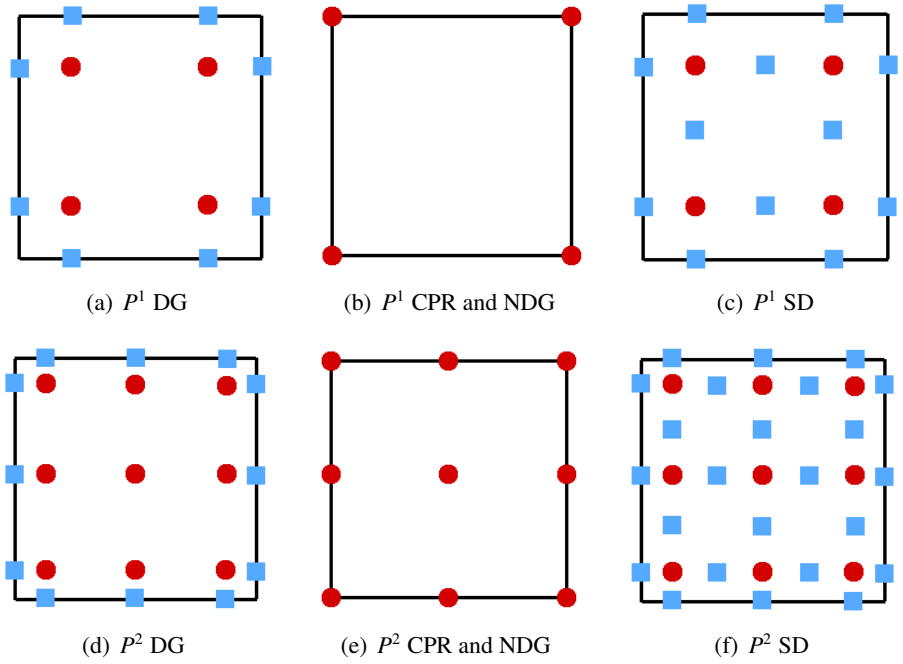
Like the CPR method, integration by parts is performed, and $\boldsymbol{q}_m$, which belongs to the space P^k, is allowed to approximate the solution on element V_m as

$$\int_{V_m}\frac{\partial \boldsymbol{q}_m}{\partial t} w dV + \int_{\partial V_m} w\vec{\boldsymbol{F}}(\boldsymbol{q}_m)\cdot\boldsymbol{n} dS - \int_{V_m}\vec{\nabla} w\cdot\vec{\boldsymbol{F}}(\boldsymbol{q}_m) dV = 0 \tag{2.1.25}$$

The solution and flux polynomials are approximated within each element m over n Gauss-Legendre points and are written as the following:

$$\boldsymbol{q}_m = \sum_{j=1}^{n}\boldsymbol{q}_{m,j}\phi_j, \quad \vec{\boldsymbol{F}}(\boldsymbol{q}_{m,j}) = \sum_{j=1}^{n}\vec{\boldsymbol{F}}_{m,j}\phi_j \tag{2.1.26}$$

where ϕ_j are the basis functions. If the basis and weighting functions are equal, then the procedure is Galerkin. The surface integral term in Equation (2.1.25) couples elements together and the common flux is again calculated via a Riemann solver. Since the solution points are Gauss-Legendre for the DG method, there is more computational work per time step when compared to the CPR method, since solutions must be interpolated to edges before element coupling. Figure 2.1 (a) and (d) show typical P^1 and P^2 DG elements, where sets of flux points are defined along the edges to communicate solutions. To increase the order of accuracy, additional solution points and flux points are defined within each element. In addition to the interpolation step, volume and surface integral calculations further increase the computational cost of the method.



(a) P^1 DG (b) P^1 CPR and NDG (c) P^1 SD

(d) P^2 DG (e) P^2 CPR and NDG (f) P^2 SD

Figure 2.1: Solution points (red circles) and flux points (blue squares) locations for P^1 and P^2 solution reconstructions. (a) and (d) Gauss-Legendre solution points with Gauss-Lobatto interface points. (b) and (e) Gauss-Lobatto solution points with coinciding flux points. (c) and (f) Gauss-Legendre solution points with Gauss-Lobatto flux points.

2.1.4 NDG Formulation

The Nodal DG formulation closely follows the CPR formulation discussed previously. Equation (2.0.1) is multiplied by a weighting function and integrated to yield the following weak form

$$\int_{V_m} \frac{\partial \boldsymbol{q}_m}{\partial t} w dV + \int_{\partial V_m} w \vec{\boldsymbol{F}}(\boldsymbol{q}_m) \cdot \boldsymbol{n} dS - \int_{V_m} \vec{\nabla} w \cdot \vec{\boldsymbol{F}}(\boldsymbol{q}_m) dV = 0 \tag{2.1.27}$$

A Riemann flux is used to apply element coupling, and replaces $\vec{\boldsymbol{F}}(\boldsymbol{q}_m) \cdot \boldsymbol{n}$ with a common Riemann flux $\boldsymbol{F}^n_{com}$, which uses the current and neighboring element information. Equation (2.1.27) is rewritten as

$$\int_{V_m} \frac{\partial \boldsymbol{q}_m}{\partial t} w dV + \int_{\partial V_m} w \boldsymbol{F}^n_{com} dS - \int_{V_m} \vec{\nabla} w \cdot \vec{\boldsymbol{F}}(\boldsymbol{q}_m) dV = 0 \tag{2.1.28}$$

A basis set, w_j, is chosen for the solution space following the DG approach, where j is the index of each solution point. Equation (2.1.28) is written in the following strong DG form as

$$\frac{\partial}{\partial t} \int_{V_m} w_i \boldsymbol{q}_{m,j} w_j dV - \int_{\partial V_m} w_i \left[\vec{\boldsymbol{F}} \cdot \boldsymbol{n} - \boldsymbol{F}^n_{com} \right] dS + \int_{V_m} w_i \vec{\nabla} \cdot \vec{\boldsymbol{F}}(\boldsymbol{q}_m) dV = 0 \tag{2.1.29}$$

Table 2.2: Derivative matrix for one SD element, $D_{i,j}$.

(i, j)	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(2,3)	(3,1)	(3,2)	(3,3)
P^1	-0.5	0.5	-	-0.5	0.5	-	-	-	-
P^2	-1.5	2.0	-0.5	-0.5	0.0	0.5	0.5	-2.0	1.5

A mass, stiffness, differentiation, and face mass matrices can now be formulated [16]. They are written as follows:

$$[M_{i,j}] = \int_{V_m} w_i w_j dV \tag{2.1.30}$$

$$[S_{i,j}] = \int_{V_m} w_i \nabla w_j dV \tag{2.1.31}$$

$$[D_{i,j}] = \left[M_{i,j}\right]^{-1} [S_{i,j}] \tag{2.1.32}$$

$$[M^A_{i,j}] = \int_{\partial V_m} w_i w_j dS \tag{2.1.33}$$

These matrices are used in Equation (2.1.29) to obtain the following formulation:

$$\frac{\partial \boldsymbol{q}_{m,j}}{\partial t} + \boldsymbol{D}\left[\vec{\boldsymbol{F}}(\boldsymbol{q}_m)\right] - \boldsymbol{L}\left[\vec{\boldsymbol{F}} \cdot \boldsymbol{n} - \boldsymbol{F}^n_{com}\right]_A = 0 \tag{2.1.34}$$

The matrix $\boldsymbol{L}$, or lifting matrix, acts on the facial degrees of freedom on face A_m. It combines the mathematical aspects of applying the mass matrix on the face, lifting the facial integral to a volume integral, and finally applying the inverse mass matrix. Much like CPR, this method also uses Gauss-Lobatto quadrature as the solution points (see Figure 2.1 (b) and (e)), simplifying the element communication step. In fact, the correction polynomials chosen for the CPR method can be selected such that they are identical to the lifting matrix terms [17]. The derivative coefficients, $[D_{i,j}]$, for P^1 and P^2 reconstructions are illustrated in Table 2.2. A '−' indicates that those indexes are not used for the reconstruction order.

2.1.5 SD Formulation

The SD scheme employs a finite-difference like approach on the conservation laws. The solution is assumed to be in the space P^k, while the flux is assumed to be in the space P^{k+1}. A set of solution points and flux points are defined within each element. Figure 2.1 (c) and (f) illustrate the point locations in a SD P^1 and P^2 element. Note how an extra flux point is required per direction for the flux polynomial.

The solution states are stored at the solution points while fluxes are stored at the flux points. Let $l(\xi)$ define the degree k Lagrange polynomial at the solution points and $h(\xi)$ be the degree $(k+1)$ Lagrange polynomial at the flux points in the ξ-direction. The coordinates (x, y) are transformed into standard coordinates (ξ, η). The $(k+1)$ solutions at the solution points build a degree k polynomial following a Lagrange polynomial basis as

$$l_j(x) = \prod_{s=1, s\neq j}^{k+1} \left(\frac{x - x_s}{x_j - x_s} \right) \tag{2.1.35}$$

where x is the point locations. In a similar manner, the flux polynomial can be built across the $(k+2)$ flux points as

$$h_{j+1/2}(x) = \prod_{s=0, s\neq j}^{k+1} \left(\frac{x - x_{s+1/2}}{x_{j+1/2} - x_{s+1/2}} \right) \tag{2.1.36}$$

In the SD method, the reconstructed solution polynomial is a tensor product of two one-dimensional polynomials of the form

$$\boldsymbol{q}(\xi, \eta) = \sum_{j=1}^{k+1} \sum_{i=1}^{k+1} \boldsymbol{q}_{i,j} l_i(\xi) l_j(\eta) \tag{2.1.37}$$

In a similar manner, the reconstructed flux polynomials are formulated as

$$\boldsymbol{f}(\xi, \eta) = \sum_{j=1}^{k+1} \sum_{i=0}^{k+1} \boldsymbol{f}_{i+1/2,j} h_{i+1/2}(\xi) l_j(\eta) \tag{2.1.38}$$

$$\boldsymbol{g}(\xi, \eta) = \sum_{j=0}^{k+1} \sum_{i=1}^{k+1} \boldsymbol{g}_{i,j+1/2} l_i(\xi) h_{j+1/2}(\eta) \tag{2.1.39}$$

In this formulation, i and j indicate the points in x and y directions respectfully. The flux polynomials are only continuous within each element. To resolve the discontinuous interface, a Riemann solver is applied at flux points on the interfaces to provide element coupling. Once the fluxes at the interface are augmented to a common value, the flux derivatives are evaluated as

$$\left(\frac{\partial \boldsymbol{f}}{\partial \xi} \right)_{i,j} = \sum_{r=0}^{k+1} \boldsymbol{f}_{r+1/2,j} h'_{r+1/2}(\xi_i) \tag{2.1.40}$$

$$\left(\frac{\partial \boldsymbol{g}}{\partial \eta} \right)_{i,j} = \sum_{r=0}^{k+1} \boldsymbol{g}_{i,r+1/2} h'_{r+1/2}(\eta_j) \tag{2.1.41}$$

The term $h'(\xi_i)$ is the derivative of the flux points Lagrange polynomial evaluated at the solution point locations ξ_i. Table 2.3 show the coefficients for interpolating the solution states to the flux points, $l_i(\xi_{j+1/2})$, and the flux derivatives evaluated at the solution points, $h'_{j+1/2}(\xi_i)$, for a P^1 reconstruction. Note that these coefficients are derived for a one-dimensional element, $[-1 \times 1]$, for Gauss-Legendre (solution) and Gauss-Lobatto (flux) points.

Table 2.3: SD coefficients for P^1 reconstruction.

(i, j)	(1,0)	(2,0)	(1,1)	(2,1)	(1,2)	(2,2)
$l_i(\xi_{j+1/2})$	1.37	0.5	-0.37	-0.37	0.5	1.37
$h'_{j+1/2}(\xi_i)$	-2.15	-2.31	0.15	-0.15	2.31	2.15

2.2 Common Numerical Functions

Each numerical method contains several generic functions outlined here, including shock capturing, element transformation, interface solvers, and time-stepping. A shock capturing scheme is needed for discontinuous problems to resolve the solution jumps and ensure numerical stability. Element transformation allows coefficients to only be computed for one element, since they will be identical for every element if the transformed elements are equivalent. A Riemann solver (interface solver) is needed to resolve solutions at element interfaces and formulate common fluxes. Finally, the time-marching scheme allows the method to advance in time.

2.2.1 Shock Capturing

To resolve solution discontinuities, the low-order and high-order methods follow two different approaches. For the FV method, the second and third order Monotonic Upstream-Centered Scheme for Conservation Laws (MUSCL) schemes are implemented, which are applied during the reconstruction of the solution at element interfaces. The slopes of the reconstructed solutions are limited with the *minmod* limiter [112]. For second order reconstruction, the second order MUSCL scheme is applied, while the third order MUSCL scheme is selected for third order reconstruction.

For the high-order methods, the same technique is applied for all schemes, which uses a *minmod* limiter (similar to FV) to find troubled elements and apply slope limiting. The updated solution is interpolated (if need be) to element edges. Once interpolation is completed, the *minmod* limiter is applied to reconstruct a second solution based on cell averaged values to the edge.

If the difference in these two values is greater than a certain threshold (ϵ) then the cell is marked for limiting, where the new solution is computed as

$$\boldsymbol{q}_{m,j} = \bar{\boldsymbol{q}}_m + (x_{m,j} - x0)\text{minmod}\left(\frac{\bar{\boldsymbol{q}}_{m+1} - \bar{\boldsymbol{q}}_m}{h}, \frac{\bar{\boldsymbol{q}}_m - \bar{\boldsymbol{q}}_{m-1}}{h}\right) \tag{2.2.1}$$

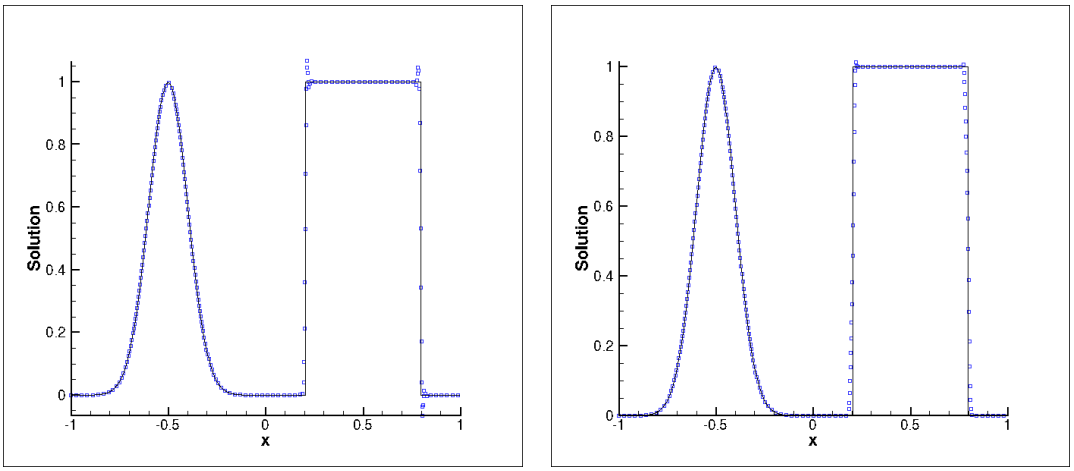
In Equation (2.2.1), $\boldsymbol{q}_{m,j}$ is the solution in element m and solution point j, $x_{m,j}$ is the location of the solution point, $x0$ is the element midpoint, h is the element size, and $\bar{\boldsymbol{q}}_m$ is the averaged solution in an element. This scheme results in a second order reconstruction which can be applied to any of the high-order methods discussed.

2.2.1.1 Shock Detector Tuning

A simple one-dimensional test problem is shown to illustrate the tuning of the ϵ parameter for shock detection. The CPR method is chosen for solution generation. The domain, $x \in [-1, 1]$, is initialized with 500 elements, with a P^2 reconstruction per element. An initial sine wave and discontinuous wave is present. The governing equation is the one-dimensional linear advection equation, given as

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0 \tag{2.2.2}$$

where u is the solution. The wave is given an initial speed of $u = 1$, and should travel two complete cycles after a time $t = 4$. Periodic boundary conditions are set to simulate an infinite domain. A third order Runge-Kutta time marching scheme (Section 2.2.4) is applied to integrate the solution through time. A small constant time step of $\Delta t = 1.0 \times 10^{-4}$ is used to minimize any temporal errors. Figure 2.2 shows four different ϵ values tested. Figure 2.2 (a) illustrates $\epsilon = 1.0$, which simulates almost no detection. The solution is shown to be oscillatory near the discontinuous solution. As the value of ϵ is decreased slightly, Figure 2.2 (b), some small oscillations are still observed near the discontinuous solution, however the smooth solution is well resolved. Further refinement shows little to no change in the computed solution. The smooth solution is still captured well, while the discontinuities are resolved without oscillations. One major drawback with the approach is the number of elements needs to be high. A coarse mesh will cause the detector to locate discontinuities in smooth regions of the domain, which will ruin the smooth solution states.



(a) $\epsilon = 1.0$

(b) $\epsilon = 1.0 \times 10^{-1}$

(c) $\epsilon = 1.0 \times 10^{-2}$

(d) $\epsilon = 1.0 \times 10^{-3}$

Figure 2.2: Shock detector parameter study. Exact solution (black line) and numerical solution (blue squares) using 1D CPR P^2 reconstruction and 500 elements. Four levels of ϵ are shown at a time of $t = 4$.

2.2.2 Element Transformation

The computational domain is composed of quadrilateral elements in two-dimensions. Each element is transformed from its standard coordinate system (x, y) to a standard quadrilateral element $(\xi, \eta) \in [-1, 1] \times [-1, 1]$. Figure 2.3 illustrates this transformation, where an element of any shape and curvature is transformed. The transformation for each element takes the form

$$\begin{pmatrix} x \\ y \end{pmatrix} = \sum_{i=1}^{N} M_i(\xi, \eta) \begin{pmatrix} x_i \\ y_i \end{pmatrix} \tag{2.2.3}$$

where N is the number of points which define the element, (x_i, y_i) are the Cartesian coordinates of those points, and $M_i(\xi, \eta)$ are the shape functions which are determined via the node locations [113]. The Jacobian matrix $\mathbf{J}$ is then given by

$$\mathbf{J} = \frac{\partial(x, y)}{\partial(\xi, \eta)} = \begin{pmatrix} x_\xi & x_\eta \\ y_\xi & y_\eta \end{pmatrix} \tag{2.2.4}$$

and when the transformation is non-singular, the inverse transformation must exist

$$\mathbf{J}^{-1} = \frac{\partial(\xi, \eta)}{\partial(x, y)} = \begin{pmatrix} \xi_x & \xi_y \\ \eta_x & \eta_y \end{pmatrix} \tag{2.2.5}$$

Additionally, the metrics can also be computed as

$$\xi_x = \frac{y_\eta}{|\mathbf{J}|} \qquad \xi_y = \frac{-x_\eta}{|\mathbf{J}|} \qquad \eta_x = \frac{-y_\xi}{|\mathbf{J}|} \qquad \eta_y = \frac{x_\xi}{|\mathbf{J}|} \tag{2.2.6}$$

Equation (2.0.1) is then transformed from the physical to the computational domain as

$$\frac{\partial \tilde{\boldsymbol{q}}}{\partial t} + \frac{\partial \tilde{\boldsymbol{f}}(\boldsymbol{q})}{\partial x} + \frac{\partial \tilde{\boldsymbol{g}}(\boldsymbol{q})}{\partial y} = 0 \tag{2.2.7}$$

where the transformed solution and flux vectors become the following:

$$\tilde{\boldsymbol{q}} = |\mathbf{J}| \boldsymbol{q} \tag{2.2.8}$$

$$\begin{pmatrix} \tilde{\boldsymbol{f}} \\ \tilde{\boldsymbol{g}} \end{pmatrix} = |\mathbf{J}| \begin{pmatrix} \boldsymbol{f}\xi_x + \boldsymbol{g}\xi_y \\ \boldsymbol{f}\eta_x + \boldsymbol{g}\eta_y \end{pmatrix} \tag{2.2.9}$$

which allows the polynomials to be constructed within standard quadrilateral elements throughout the domain.

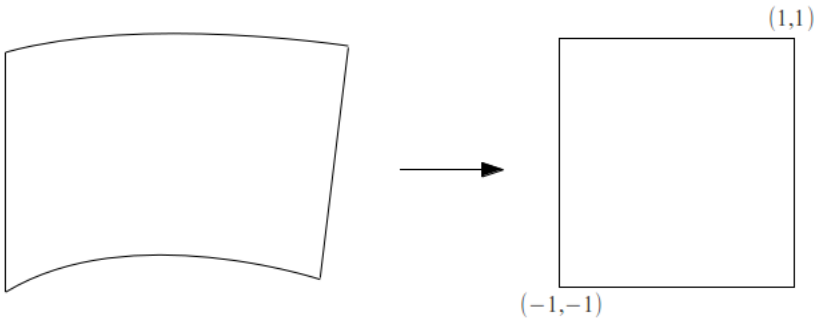


Figure 2.3: Transformation from a physical element to the standard element.

2.2.3 Element Coupling

In general, the neighboring states along element interfaces are not necessarily equivalent. Along an edge, a left and right solution state exist, where

$$\boldsymbol{q}_L = \begin{pmatrix} \rho_L \\ \rho_L u_L \\ \rho_L v_L \\ e_L \end{pmatrix} \neq \begin{pmatrix} \rho_R \\ \rho_R u_R \\ \rho_R v_R \\ e_R \end{pmatrix} = \boldsymbol{q}_R \tag{2.2.10}$$

This leads to solving a Riemann problem to compute the continuous solution over the interfaces. Two Riemann solvers are presented in this thesis: The Rusanov and HLLC solvers.

2.2.3.1 Rusanov flux

The Rusanov flux [114] requires the computation of averaged pressure, density, and speed of sound:

$$\bar{p} = \frac{p_L + p_R}{2} \tag{2.2.11}$$

$$\bar{\rho} = \frac{\rho_L + \rho_R}{2} \tag{2.2.12}$$

$$\bar{a} = \sqrt{\frac{\gamma \bar{p}}{\bar{\rho}}} \tag{2.2.13}$$

where the speed of sound computation is for an ideal gas equation of state. At the interface, the average speed of the flow in the normal direction is computed as

$$\bar{v}_n = \frac{|\boldsymbol{V}_L \cdot \boldsymbol{n} + \boldsymbol{V}_R \cdot \boldsymbol{n}|}{2} \tag{2.2.14}$$

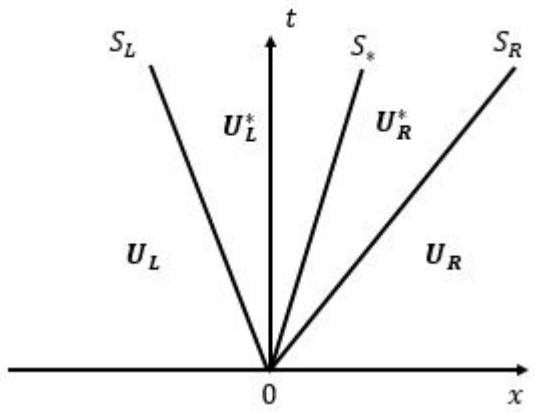


Figure 2.4: HLLC approximate Riemann solver. The star region solutions consists of two states separated by the middle wave speed S_*.

where $\boldsymbol{V}_L$ is the velocity vector of the left state and $\boldsymbol{n}$ is the normal vector at the interface. The Rusanov flux is then computed using a combination of the left and right states with the averaged flow speeds as

$$\vec{\boldsymbol{F}}^n_{rus} = \frac{1}{2}\left[\vec{\boldsymbol{F}}(\boldsymbol{q}_R)\cdot\boldsymbol{n} + \vec{\boldsymbol{F}}(\boldsymbol{q}_L)\cdot\boldsymbol{n}\right] - \frac{1}{2}(\bar{v}_n + \bar{a})(\boldsymbol{q}_R - \boldsymbol{q}_L) \tag{2.2.15}$$

The Rusanov flux is applied in cases without material interfaces, and is the interface solver implemented when comparing the different numerical methods in Chapter 4.

2.2.3.2 HLLC flux

The HLLC scheme is a modification of the Harten, Lax, and van Leer (HLL) scheme [115], developed by Toro, Spruce, and Speares [116, 117]. One shortcoming of the Rusanov and HLL solvers is that they do not factor intermediate waves in the formulations. Figure 2.4 shows the approximate Riemann solver using the HLLC scheme. In the typical Riemann problem, all that matters is the average states across the wave structure, with no regard for any variations in the star regions [118]. It was found by Harten, Lax, and Van Leer that this defect can be corrected by restoring the missing waves [115]. This plays an important factor when dealing with material interfaces.

In the HLLC solver, the missing middle waves are restored into the structure of the Riemann solver. The HLLC approximate Riemann solver is given as

$$\hat{\boldsymbol{U}}(x,t) = \begin{cases} \boldsymbol{U}_L \text{ , if } \frac{x}{t} \leq S_L \\ \boldsymbol{U}_L^* \text{ , if } S_L \leq \frac{x}{t} \leq S_* \\ \boldsymbol{U}_R^* \text{ , if } S_* \leq \frac{x}{t} \leq S_R \\ \boldsymbol{U}_R \text{ , if } \frac{x}{t} \geq S_R \end{cases} \tag{2.2.16}$$

where the corresponding flux is

$$\boldsymbol{F}_{hllc}(x,t) = \begin{cases} \boldsymbol{F}_L \text{ , if } 0 \leq S_L \\ \boldsymbol{F}_L^* \text{ , if } S_L \leq 0 \leq S_* \\ \boldsymbol{F}_R^* \text{ , if } S_* \leq 0 \leq S_R \\ \boldsymbol{F}_R \text{ , if } 0 \geq S_R \end{cases} \tag{2.2.17}$$

The formulation presented follows the notation that the left and right states, $\boldsymbol{U}_L$ and $\boldsymbol{U}_R$, are vectors of the states and are identical to $\boldsymbol{q}_L$ and $\boldsymbol{q}_R$. The intermediate flux values, denoted by the stars, need to be computed. The left and right wave speeds (S_L, S_R) are assumed to be known, and can be computed using any type of wave-speed estimates. In this thesis, the Davis approximation [119] is used, where the speeds are computed as

$$S_L = u_L - a_L \tag{2.2.18}$$

$$S_R = u_R + a_R \tag{2.2.19}$$

where u is the fluid velocity and a is the speed of sound. The star region wave speed, S_*, can be computed in terms of the left and right solution states and the wave speed estimates as

$$S_* = \frac{\rho_L u_L (S_L - u_L) - \rho_R u_R (S_r - u_R) + p_R - p_L}{\rho_L (S_L - u_L) - \rho_R (S_R - u_R)} \tag{2.2.20}$$

With all three wave speeds computed, the appropriate location can be found using Equation (2.2.17). The fluxes F_L and F_R can be computed from Equation (2.0.4). The fluxes in the star regions are computed as the following:

$$\boldsymbol{F}_L^* = \boldsymbol{F}_L + S_L(\boldsymbol{U}_L^* - \boldsymbol{U}_L) \tag{2.2.21}$$

$$\boldsymbol{F}_R^* = \boldsymbol{F}_R + S_R(\boldsymbol{U}_R^* - \boldsymbol{U}_R) \tag{2.2.22}$$

Now, the only remaining required computations are $\boldsymbol{U}^*$. The star region states are determined using the left or right states, depending on the region. The formulation is written as

$$\boldsymbol{U}_K^* = \frac{S_K\boldsymbol{U}_K - \boldsymbol{F}_K + p_K^*\boldsymbol{D}_*}{S_K - S_*} \tag{2.2.23}$$

In Equation (2.2.23), K is the index for either left (L) or right (R) states, p_K^* are the star region pressures, and $\boldsymbol{D}^*$ is a constant matrix. The pressures are computed as follows:

$$p_L^* = p_L + \rho_L(S_L - u_L)(S_* - u_L) \tag{2.2.24}$$

$$p_R^* = p_R + \rho_R(S_R - u_R)(S_* - u_R) \tag{2.2.25}$$

while the constant matrix is defined as

$$\boldsymbol{D}^* = \begin{pmatrix} 0 \\ 1 \\ 1 \\ S_* \end{pmatrix} \tag{2.2.26}$$

for the two-dimensional Euler system. There are several formulations for the HLLC solver, such as enforcing the condition $p_L^* = p_R^*$. In the form presented here, this condition is relaxed and is more consistent with pressure approximations [118].

2.2.4 Time-Stepping

Explicit time integration is completed using a three-stage Runge-Kutta [120] scheme to march the solution forward in time from t to $t + 1$ as

$$\boldsymbol{q}^{(1)} = \boldsymbol{q}^t + \Delta t\boldsymbol{r}(\boldsymbol{q}^t) \tag{2.2.27}$$

$$\boldsymbol{q}^{(2)} = \frac{3}{4}\boldsymbol{q}^t + \frac{1}{4}\boldsymbol{q}^{(1)} + \frac{1}{4}\Delta t\boldsymbol{r}(\boldsymbol{q}^{(1)}) \tag{2.2.28}$$

$$\boldsymbol{q}^{(t+1)} = \frac{1}{3}\boldsymbol{q}^t + \frac{2}{3}\boldsymbol{q}^{(2)} + \frac{2}{3}\Delta t\boldsymbol{r}(\boldsymbol{q}^{(2)}) \tag{2.2.29}$$

where Δt is the chosen time step for integration and $\boldsymbol{r}(\boldsymbol{q}^n)$ is the right hand side of the system evaluated at step level n.

2.2.5 CFL Computation

To conduct a fair comparison between all the methods, each method should be allowed to take its maximum allowable time-step. The time-step for each method is computed using the CFL condition as

$$\Delta t \leq \frac{CFL\Delta x}{|u| + c} \tag{2.2.30}$$

where CFL is the Courant-Friedrichs-Lewy condition, Δx is the element size, $|u|$ is the velocity magnitude, and c is the wave speed computed using

$$c = \sqrt{\frac{\gamma p}{\rho}} \tag{2.2.31}$$

for the ideal gas equation of state. For high-order methods, the time-step is computed at each point and the maximum allowable time-step is found by comparing all time-steps at all points. For the FV method, the time-steps are computed along element edges, and again the maximum time-step is found by comparing all time-steps along the edges.

CHAPTER 3. GPU IMPLEMENTATION

Graphics processing units (GPUs) are mostly used for graphics acceleration, computing images shown on a computer screen. This image rendering task is largely parallel, which the GPU architecture takes advantage of. Recently, scientific computing utilizing GPUs is becoming more appealing, especially under NVIDIA's Compute Unified Device Architecture (CUDA). The GPU hardware and CUDA capabilities are continuously updated to improve performance. This chapter focuses on the overall GPU CUDA model and implementation Fof the numerical methods in the GPU's architecture.

3.1 CUDA Overview

GPUs are built around streaming multiprocessors (SMs) to complete tasks, which execute hundreds of independent threads. The multiprocessors launch blocks, containing threads, running in parallel. Before discussing the threads and blocks, a small introduction into SMs is presented. A GPUs SM count determines the number of tasks which can be completed in parallel. Multiple tasks can be executed concurrently on one multiprocessor, and once the tasks are complete, new tasks are launched on the multiprocessor. It follows that a GPU with a high SM count can complete many tasks in parallel, and will complete the overall program faster than a GPU with a low SM count.

A GPU is composed of grids, blocks, and threads. Each GPU function (called a kernel) forms a grid which has a specified amount of blocks. Within each block, a set number of threads are initialized to run. A multiprocessor executes hundreds of threads at once, which it completes using a Single-Instruction, Multiple-Thread (SIMT) architecture. In SIMT architecture, the SM executes threads in groupings of 32, called warps. When a SM needs to run threads across multiple blocks, the threads are partitioned into warps and scheduled for execution. Each warp executes one common instruction at a time, so if all 32 threads have an identical execution path, high computational efficiency is obtained. However, if the

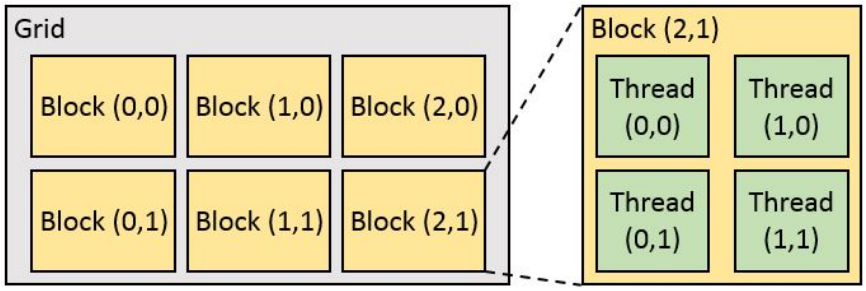


Figure 3.1: CUDA 2D grid and block illustration.

warp threads diverge due to a conditional branch, the branches are executed serially. The threads converge once the different paths are completed, thus it is ideal to construct problems around warps. Since a SM contains multiple blocks and multiple threads, an addressing scheme is needed to access the individual threads and blocks. Each SM can have blocks in two dimensions (gridDim.x, gridDim.y), while each block can have threads in three dimensions (blockDim.x, blockDim.y, blockDim.z). Each individual block index can be accessed (blockIdx.x, blockIdx.y) as can each individual thread (threadIdx.x, threadIdx.y, threadIdx.z). This enables programmers to have specific threads and blocks access specific data locations in memory. Figure 3.1 shows a two-dimensional grid which contains two-dimensional blocks. In this illustration, (gridDim.x, gridDim.y) = $(3, 2)$ and (blockDim.x, blockDim.y, blockDim.z) = $(2, 2, 0)$. Each block is referenced using a block index, where in the example, blockIdx.x = 0, 1 or 2, and blockIdx.y = 0 or 1. The threads are also indexed and following the example, threadIdx.x = 0 or 1 and threadIdx.y = 0 or 1. This yields a kernel with 24 threads.

The GPU has multiple memory types to utilize for specific needs. All threads and all blocks can see the GPUs global memory and can write or read from the memory. However, write access to this memory can be high (hundreds of clock cycles), hence writes should be completed only when necessary. For memory access, if the read is coalesced (neighboring threads access neighboring memory locations) then fast memory access is achieved. The global memory can be bound into texture memory, which is read-only and cached. Texture memory is optimized for spatial locality and benefits from warps accessing nearby texture locations. Each thread has its own private memory, located in registers and local space. Small arrays and variables will be put into registers, where calculations are completed very fast (10 to 20 clock cycles) [3]. The size of registers is not infinite, and if too much register space is demanded, data spills into local memory. The local memory access is equivalent to global memory, so

care should be taken when utilizing registers. The final memory to consider is shared memory, which has a higher bandwidth than local or global memory. Shared memory is useful if accessing data more than once with the same thread or from different threads within a block. However, shared memory in one block cannot be seen by the threads in another block, and its lifetime is the lifetime of the block.

There are a few rules to follow when writing CUDA code to help optimize computing speed.

- The usage of shared memory should be minimized and reused when possible.
- The storage locations of memory should compliment the SIMT architecture.
- Threads should be synchronized rarely and in optimal locations.
- Each thread should write to global memory only once.

Some are quite obvious, such as the recycling of shared memory and location of barriers (called synchronization with CUDA). For storage order of memory, consider the following case: Let's use the Euler system and assume a memory storage where at a single point, memory position 0 is conservation of mass, memory position 1 and 2 are conservation of momentum in x and y, and memory position 3 is conservation of energy. Now, let thread 0 read memory position 0, thread 1 read memory position 1, and so on. Hence, the threads in the warp are evaluating different expressions, which means different code, inefficient for SIMT architecture. A better solution is to let thread 0 access memory position 0 of the point, and thread 1 access the memory position 0 of another point, which allows the same expression to be computed by the threads. The final item, one global write, is also self explanatory, since each access to global memory is expensive. It is noted, however, that in some cases this cannot be followed, and allowing multiple writes to global is cheaper than splitting the algorithm into multiple kernels.

Some conventions are now listed to simplify the algorithms presented, and assist the reader. Threads and blocks are allowed to be multi-dimensional, and have the indexes t_x, t_y, t_z, b_x, and b_y (threads can have three indexes while blocks can have up to two). Memory locations are presented in the following manner: If the variable has no superscript, it resides in the local memory to the thread (typically the registers). The other three locations are denoted by superscripts g, t, and s to represent global, texture, and shared memory space respectfully. In addition, any memory reads or writes with indexes will be denoted in the following manner: If stored memory needs to be read (say c is the pointer or array which

holds the information), and the indexes depend on i, j, and k, then let $v = c[(i, j, k)]$. Meaning, v now reads information in c at an index location which depends on i, j, and k (not a three dimensional array or pointer).

3.2 CUDA Implementation

Now that the basic idea of CUDA is introduced, the GPU implementation of each method is presented. An overview of each method's steps are outlined in Figure 3.2 (omitting some common algorithms). For each method, the local and non-local operations are shown. A local operation means all information to complete the operation is contained within the element, while non-local means communication must occur between elements. Note that the number of operations listed does nor correlate with the number of functions required. Some operations, local and non-local, can be combined into one function to reduce memory loading and multiple sweeps through the domain. Not shown are algorithms to copy the solution array into global memory and the time marching scheme. Both algorithms will be presented first.

3.2.1 General Kernels

Several generic CUDA kernels are presented which are common between all the methods, with minor adjustments depending on the method. First, the solution needs to be copied into a storage location for use with the time-stepping algorithm. The *GPU_Copy* kernel completes this task. The block grid is defined as $\vec{t} = \left[n_{sp}, n_v, \frac{64}{n_{sp}}\right]$, while the grid is $b = \left[\left(\frac{n_e}{t_z} * \frac{1}{SM_c} + 1\right) * SM_c\right]$. The terms are as follows: n_{sp} is the number of solution points per element ($n_{sp} = 1$ for FV), n_v is the number of state variables, n_e is the number of elements, and SM_c is the number of shared multiprocessors. On the cards considered in this work, $SM_c = 13$. The value of 64 was selected from testing variable numbers from 32 to 512, multiples of 32. As an example calculation, consider a mesh with 2,000 elements and 9 points per element with 4 states. The block grid becomes $\vec{t} = [9, 4, 7]$, since $\frac{64}{9} = 7$ rounded down (integer divisions). The corresponding grid is $b = \left[\left(285 * \frac{1}{13} + 1\right) * 13\right] = [286]$. A grid of 286 blocks are scheduled, where each block runs 252 threads, scheduled in groupings of warps. The *GPU_Copy* kernel copies solution data from one global memory array to another. Since neighboring threads are

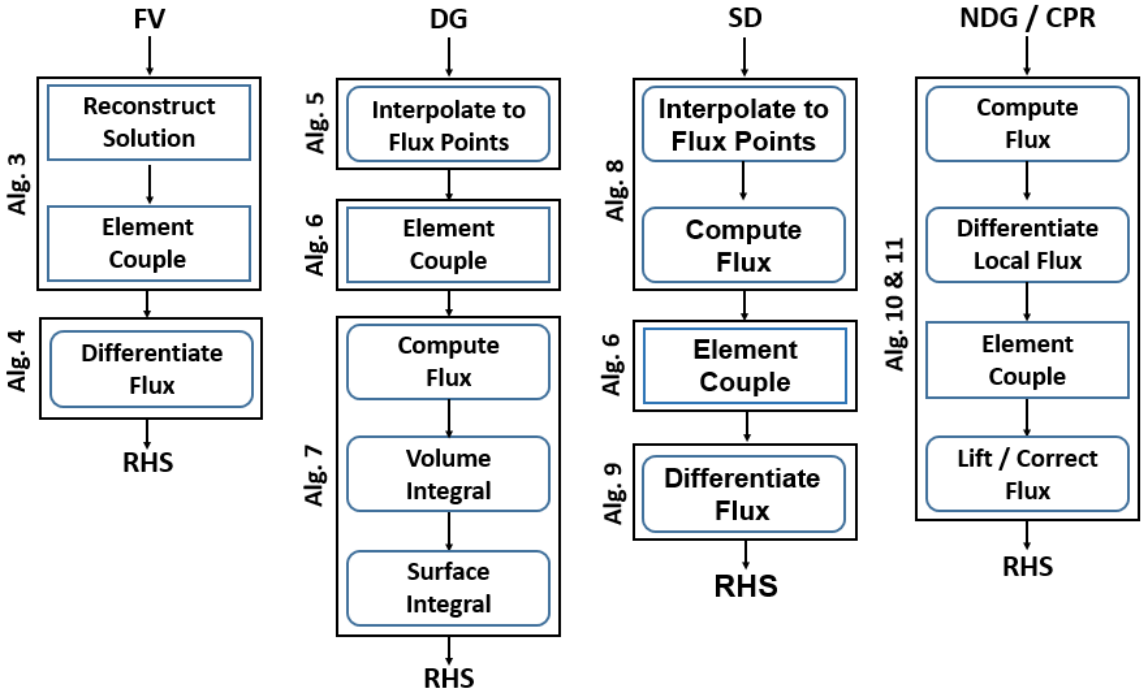


Figure 3.2: Overview of methods. Blue rectangles indicate non-local operations, while curved edges indicate local operations. Surrounding black boxes show kernel groupings by algorithms.

Algorithm 1 GPU_Copy

▷ Solution point in a element
t_x = threadIdx.x
▷ State vector
t_y = threadIdx.y
▷ Current element in the block
tmp = threadIdx.z
▷ Current global element
k = blockIdx.x $*$ blockDim.z + tmp
if $k < \mathrm{n}_e$ **then**
    ▷ Copy the solution
    $q_o^g[(t_x, t_y, k)] = q^g[(t_x, t_y, k)]$
end if

accessing neighboring cells in memory, the access is coalesced and fast. No other optimizations or changes are required.

The time-stepping kernel, *GPU_RK,* updates the solution in time and is completed in a very similar manner to the *GPU_Copy* kernel. Once again, all access is in global memory, with several inputs to

Algorithm 2 GPU_RK(a,b,Δt)

▷ Solution point in a element
t_x = threadIdx.x
▷ State vector
t_y = threadIdx.y
▷ Current element in the block
tmp = threadIdx.z
▷ Current global element
k = blockIdx.x $*$ blockDim.z + tmp
if $k < \mathrm{n}_e$ **then**
    ▷ Update in time
    $q^g[(t_x, t_y, k)] = a * q_o^g[(t_x, t_y, k)] + (1 - a) * q^g[(t_x, t_y, k)] + b * \Delta t * r^g[(t_x, t_y, k)]$
end if

complete the necessary computations, including the time step, Δt, and coefficients a and b which depend on what update stage is being completed. The access of memory is coalesced in a similar manner to the *GPU_Copy* kernel.

3.2.2 FV CUDA

The FV method can be separated into two kernels to update the residual. As shown in Figure 3.2, one kernel reconstructs the solution and provides element coupling (both non-local operations) whose output feeds into the flux differentiation kernel.

The *FV_Reconstruct* algorithm is outlined in Algorithm 3, which reconstructs the left and right solutions at faces and computes the Riemann flux at the face. The implementation uses strictly texture memory and registers, with n_v writes to global memory per thread to finish the algorithm. The threads are defined as faces, t_x, in the domain. In all algorithms, multiple elements (or faces) are calculated in one block, increasing the parallelism of the algorithm. The variables n_v and n_e denote the number of state variables and number of elements respectfully. Depending on the degree of the reconstruction polynomial, an appropriate amount of information from neighbors is loaded (the index e_1 denotes ele-

Algorithm 3 FV_Reconstruct

▷ Faces in element
t_x = threadIdx.x
▷ Current global face
k = blockIdx.x $*$ blockDim.x + threadIdx.y
if $k < \mathrm{n}_e$ **then**
  ▷ Gather information from neighbors
  $q_{e_1}[(0...n_v)] = q^t[id_{e_1}(t_x, k)]$
  $q_{e_2}[(0...n_v)] = q^t[id_{e_2}(t_x, k)]$
  ...
  ▷ Reconstruct left and right solutions
  $q_L[(0...n_v)] = f(q_{e_1}, q_{e_2}...)$
  $q_R[(0...n_v)] = f(q_{e_1}, q_{e_2}...)$
  ▷ Compute the interface flux
  InterfaceFlux(q_L, q_R, f_n)
  ▷ Store interface flux into global memory
  $f^g[k + (0...n_v) * n_f] = f_n[(0...n_v)]$
end if

ment 1). Once the data is loaded, the appropriate reconstruction formula (see Section 2.1.1) is applied and the flux at the interface is computed and stored.

To compute the flux derivative, multiple elements are computed per block, and each thread reads the appropriate flux information from texture memory to compute the flux derivative in the element (computed from *FV_Reconstruct*). The algorithm *FV_Flux* shows the residual update. The threads run over the solution states and the local element within the block. Each thread reads the inverse length of the element into registers from texture memory. Then, each thread reads the flux data on each face in the element. Again, the read is from texture memory into registers. Once all memory reads are completed, the residual is computed from registers and stored into global memory.

3.2.3 DG CUDA

As shown in Figure 3.2, the decomposition of the DG residual update requires three kernels. The three algorithms interpolate the information from solution points to flux points on element edges, couple the elements via a Riemann flux, and compute the volume and surface integrals using information stored at both solution and flux points.

Algorithm 4 FV_Flux

▷ Current solution state
t_x = threadIdx.x
▷ Current global element
k = blockIdx.x ∗ blockDim.x + threadIdx.y
if $k < \mathrm{n}_e$ **then**
  ▷ Read cell edge inverse length info
  $dx[(0...1)] = dx^t[(0...1), k]$
  $dy[(0...1)] = dy^t[(0...1), k]$
  ▷ Read in flux data from texture
  $f_x[(0...1, t_x)] = f_x^t[(0...1, t_x, k)]$
  $f_y[(0...1, t_x)] = f_y^t[(0...1, t_x, k)]$
  ▷ Update residual
  $r^g[(t_x, k)] = (dx[0] * f_x[0]) + (dx[1] * f_x[1]) + (dy[0] * f_y[0]) + (dy[1] * f_y[1])$
end if

The *DG_Interpolation* kernel, shown in Algorithm 5, runs threads along each point in all the faces in the domain. The n_{sp1d} variable is the number of solution points in a one-dimensional line. At each face, the solution point information is read from texture memory, which serves as an index to read the required state at the solution points. The solution states and interpolation coefficients, *cint*, are read from textured memory to perform the indicated operation, which is stored in global memory for future access. One might consider utilizing the GPUs shared memory for such an operation. Reading in the interpolation coefficients into shared memory would yield an inefficient algorithm, as the data access to the memory per thread is only once, and the information is not shared between other threads. For the solution states, a single memory location is accessed four times (interpolation over four faces). The required shared memory read and thread synchronization prior to the interpolation loop does not yield improved speed with such minimal memory reads.

To couple the elements, the *DG_Couple* kernel (Algorithm 6) only requires the left and right information at interfaces, obtained from the *DG_Interpolation* kernel. The threads run over flux points on element faces, and each thread reads the required face normal, area, and cell indexes. A conditional branch is required to enforce boundary conditions where required. This yields thread divergence, and serial execution is completed until the branch is closed and the threads converge. Once the threads converge, the interface flux is built into registers and written into global memory space.

Algorithm 5 DG_Interpolation

▷ Point on element face
t_x = threadIdx.x
▷ Current face in block
t_y = threadIdx.y
▷ The global face
f = blockIdx.x ∗ blockDim.y + t_y
if $f < \mathrm{n}_f$ **then**
  ▷ Gather index information on face
  $m = id^t_f(t_x, f)$
  for $l = 0$ to n_{sp1d} **do**
    ▷ Read solution point information and operate
    $id = id^t_{sp}[(m, l)]$
    $q_l[(0...n_v)] = q_l[(0...n_v)] + cint^t[l] * q^t[id]$
  end for
  $q^g_l[(t_x, f, 0...n_v)] = q_l[(0...n_v)]$
end if

The final algorithm for DG, *DG_Flux* illustrated in Algorithm 7, updates the flux derivatives. The threads run on solution points within elements, and multiple elements are packed within a thread block. A sufficient amount of shared memory is allocated for flux storage, and threads are halted while the memory is loaded. Shared memory allocation is required on a per block basis. Consider a thread grid $\vec{t} = \left[n_{sp}, \frac{B_{base}}{n_{sp}}\right]$, where B_{base} is the block base, typically multiples of 32. Table 3.1 illustrates the allocation size of shared memory based on the block base and the reconstruction order. In addition, the amount of blocks computed in one SM, B_{SM}, is shown. The shared memory size needs to be hard-coded into the algorithm, so minimizing the allocation cost when switching orders of accuracy is ideal. Comparing the shared memory allocation costs, there is only a difference of 4 with a 64 block base between the reconstruction orders. The difference increases with increasing block base. With a block base of 32 (not shown) a difference of 20 is observed. From the table, the ideal block base is 64, which minimizes the amount of wasted memory space. In the *DG_Flux* algorithm, the solution is read from global memory and used to compute the flux terms, which is stored in shared memory. When loading information into shared memory, the threads in the block need to wait for all threads to complete the memory load, hence the *syncthreads* command is issued. Shared memory in this case offers high computational efficiency, since the volume integration for one solution point requires information at all other solution points in one element. The volume integration calculation is a function (V) of the stiffness

Algorithm 6 DG_Couple

▷ Point on element face
t_x = threadIdx.x
▷ Current face in block
t_y = threadIdx.y
▷ The global face
f = blockIdx.x ∗ blockDim.y + t_y
if $f < \mathrm{n}_f$ **then**
    ▷ Read data (normals, cell indexes)
    ...
    ▷ Conditional branch for boundary conditions
    ...
    ▷ Read in left and right solution
    $q_L[(0...n_v)] = q^t[(id_{e_1}(t_x, f), 0...n_v)]$
    $q_R[(0...n_v)] = q^t[(id_{e_2}(t_x, f), 0...n_v)]$
    ▷ Compute the interface flux
    InterfaceFlux(q_L, q_R, f_n)
    ▷ Store interface flux into global memory
    $f_n^g[(t_x, f, 0...n_v)] = f_n[(0...n_v)]$
end if

matrix coefficients and the fluxes, which is summed at each point. The surface integral is computed in a similar manner, but texture memory is used instead of shared memory. The flux values along the face edges are read using textures (data spatial locality) and operate with the integration term to compute the surface integral in registers. The flux derivative is assembled and stored in the GPUs global memory.

3.2.4 SD CUDA

Like DG, SD also decomposes nicely into three separate kernels as shown in Figure 3.2: Interpolations, coupling, and flux computation. Two major differences in implementation are the following: SD has no volume integration and each element has interior flux points (not just on the edges). This aspect makes the interpolation more expensive in terms of operations and storage, but the final flux evaluation cheaper. One important computational aspect implemented with the SD method on CUDA is *thread switching*, where the threads in one kernel switch from operating on flux points to solution points. The switch is primarily useful for loading information into shared memory.

The interpolation must be completed in each coordinate direction separately, and both the solution states and flux terms must be stored in global memory for future use. Algorithm 8 shows the

Algorithm 7 DG_Flux

▷ Solution points, current element in block, global block
t_x = threadIdx.x
t_y = threadIdx.y
k = blockIdx.x ∗ blockDim.y + t_y
▷ Allocate shared memory space
__shared__ $f^s[size]$
__shared__ $g^s[size]$
if $k < \mathrm{n}_e$ **then**
  ▷ Read state at solution points
  $q[(0...n_v)] = q^g[(t_x, k, 0...n_v)]$
  ▷ Compute flux into shared memory
  $f^s[t_x, t_y, (0...nv)] = F(q[(0...n_v)])$
  $g^s[t_x, t_y, (0...nv)] = G(q[(0...n_v)])$
  ▷ Threads need to wait for shared memory to fill
  syncthreads()
  ▷ Compute volume integral using shared memory
  for $l = 0$ to n_{sp} **do**
    ▷ Stiffness matrix coefficients
    $(S_x, S_y) = (S_x^t, S_y^t)[(t_x, l)]$
    $Vol[(0...n_v)] = Vol[(0...n_v)] + V([S_x, S_y, f^s, f^y])$
  end for
  ▷ Surface integral next
  for $l = 0$ to n_{fp} **do**
    ▷ Read integration term
    $I = I^t[(t_x, l)]$
    ▷ Read in flux at interface points and compute surface integral
    $f_n[(0...n_v)] = f_n^t[(l, k, 0...n_v)]$
    $Sur[(0...n_v)] = Sur[(0...n_v)] - f_n[(0...n_v)] * I$
  end for
  ▷ Assemble flux derivative and store
  $r^g[(t_x, k, 0...n_v)] = M^{-1}[t_x] * (Vol[(0...n_v)] + Sur[(0...n_v)])$
end if

Algorithm 8 SD_Interpolation

▷ Flux points in one direction ($n_{sp1d} * n_{fp1d}$)
t_x = threadIdx.x
▷ Current element in block
t_y = threadIdx.y
▷ The global element
k = blockIdx.x * blockDim.y + t_y
▷ Solution and flux point indexes
$isp = mod(t_x, n_{sp1d})$
$i_{fp} = t_x / n_{sp1d}$
▷ Shared memory allocation
__shared__ $q^s[size]$
if $k < \mathrm{n}_e$ **then**
  ▷ Read states into shared memory
  if $i_{fp} < \mathrm{n}_{sp1d}$ **then**
    $q^s[(i_{sp}, i_{fp}, t_y, 0...n_v)] = q^t[i_{sp}, i_{fp}, k, 0...n_v)]$
  end if
  syncthreads()
  ▷ Build polynomial at flux points (x-direction)
  for $l = 0$ to n_{sp1d} **do**
    $q_x[(0...n_v)] = q_x[(0...n_v)] + cint^t[(l, i_{fp})] * q^s[(l, i_{sp}, t_y, 0...n_v)]$
  end for
  ▷ Compute flux terms
  $f_x[(0...nv)] = F(q_x[(0...n_v)])$
  ▷ Store states and flux at flux points
  $q^g_{x,y}[(i_{sp}, i_{fp}, k, 0...n_v)] = q_x[(0...n_v)]$
  $f^g_{x,y}[(i_{sp}, i_{fp}, k, 0...n_v)] = f_x[(0...n_v)]$
  ▷ Build polynomial at flux points (y-direction)
  for $l = 0$ to n_{sp1d} **do**
    $q_y[(0...n_v)] = q_y[(0...n_v)] + cint^t[(l, i_{fp})] * q^s[(l, i_{sp}, t_y, 0...n_v)]$
  end for
  ▷ Compute flux terms
  $f_y[(0...nv)] = F(q_y[(0...n_v)])$
  ▷ Store states and flux at flux points
  $q^g_{x,y}[(i_{sp}, i_{fp}, k, 0...n_v)] = q_y[(0...n_v)]$
  $f^g_{x,y}[(i_{sp}, i_{fp}, k, 0...n_v)] = f_y[(0...n_v)]$
end if

Table 3.1: Shared memory requirements: *DG_Flux*

P^k	n_{sp}	B_{base}	B_{SM}	Size	Bytes
P^1	4	64	16	256	2,048
-	-	128	32	512	4,096
-	-	256	64	1024	8,192
P^2	9	64	7	252	2,016
-	-	128	14	504	4,032
-	-	256	28	1008	8,064

SD_Interpolation kernel which has a thread grid of $\vec{t} = \left[n_{sp1d} * n_{fp1d}, \frac{B_{base}}{n_{sp1d} * n_{fp1d}}\right]$, where $n_{sp1d} * n_{fp1d}$ is the total number of flux points in one direction. For P^1 and P^2 reconstructions, this term equals 6 and 12 respectfully. The interpolation kernel uses modular arithmetic and integer divisions to obtain the indexes of solution points and flux points in the direction (i_{sp}, i_{fp}). These indexes are used to control which points to operate with. The flux point index is first restricted to allows the threads to operate on solution points and read in the solution states to shared memory. Note that for SD, $n_{sp} < n_{fp}$, regardless of order of accuracy. The shared memory will be used for both interpolation in the x and y-directions. The data interpolation uses coefficients from textured memory, *cint*, and shared memory. Note that the *cint* presented in this algorithm is not equivalent to the *cint* in Algorithm 5 (*DG_Interpolation*). Once the interpolation is completed, the solution states are available on the flux points at element edges, which allows the elements to be coupled. The coupling kernel is nearly identical to the *DG_Couple* algorithm and is omitted. The only difference appears when writing the flux information into global memory. The *SD_Interpolation* kernel already writes information into the global flux memory, even though this is not the coupled flux. This memory is over-written in the coupling algorithm, which may bring a sense of inefficiency. However, in this special case, overwriting the memory is more efficient. To prohibit the memory over-write, the interpolation kernel would require threads to diverge when writing the flux terms, since the solution states are also required on the element edges. A thread divergent free kernel is significantly better, as divergence yields serial computations. A second option is to split the kernel in two, one kernel to operate with the flux, and another to operate with the solution states.

However, this requires the same memory to be loaded multiple times and a second sweep through the domain. For these reasons, extra memory writes in one kernel yield an efficient algorithm.

Algorithm 9 SD_Flux

▷ Flux points in one direction ($n_{sp1d} * n_{fp1d}$)
t_x = threadIdx.x
▷ Current element in the block and global element
t_y = threadIdx.y
k = blockIdx.x * blockDim.y + t_y
▷ Solution point and flux point indexes
$isp = mod(t_x, n_{sp1d})$
$ifp = t_x / n_{sp1d}$
▷ Shared memory allocation
__shared__ $f_x^s[size]$
__shared__ $f_y^s[size]$
if $k < \mathrm{n}_e$ **then**
  ▷ Read fluxes into shared memory
  $id_x = id_x^t(isp, ifp)$
  $id_y = id_y^t(isp, ifp)$
  $f_x^s[(id_x, t_y, 0...n_v)] = f_{x,y}^t[(id_x, k, 0...n_v)]$
  $f_y^s[(id_y, t_y, 0...n_v)] = f_{x,y}^t[(id_y, k, 0...n_v)]$
  syncthreads()
  ▷ Now only run on solution points
  if $ifp < n_{sp1d}$ **then**
    ▷ Flux differentiation on solution points
    for l = 0 to n_{fp1d} **do**
      ▷ Derivative coefficients
      $c_x = c_x[(isp, l)]$
      $c_y = c_y[(ifp, l)]$
      ▷ Flux derivative per direction
      $dF_x[(0...n_v)] = dF_x[(0...n_v)] + c_x * f_x^s[(id_x, l, t_y, 0...n_v)]$
      $dF_y[(0...n_v)] = dF_y[(0...n_v)] + c_y * f_y^s[(id_y, l, t_y, 0...n_v)]$
    end for
    $r^g[(t_x, k, 0...n_v)] = dF_x[(0...n_v)] + dF_y[(0...n_v)]$
  end if
end if

The kernel *SD_Flux* (Algorithm 9) gathers all the flux terms and computes the derivative at the solution points. The threads are allowed to run across solution and flux points in one direction. This allows the flux to be loaded into shared memory in the two coordinate directions (x, y). The shared memory required is shown in Table 3.2 for P^1 and P^2 reconstructions. Due to the nature of the method,

Table 3.2: Shared memory requirements: *SD_Flux*

P^k	$n_{sp1d} * n_{fp1d}$	B_{base}	B_{SM}	Size	Bytes
P^1	6	64	16	384	3,072
-	-	128	32	768	6,144
-	-	256	64	1536	12,288
P^2	12	64	7	336	2,688
-	-	128	14	672	5,376
-	-	256	28	1344	10,752

the allocation size and difference varies significantly from the DG requirements in Table 3.1. The extra interior flux points increase the amount of stored shared memory. Due to the large variations in allocation amount, a block base of 64 was chosen for the method. After the threads have loaded the shared memory in each coordinate direction, the threads are synchronized and the threads switch to operate on solution points. Implementations of the method were performed without shared memory, and threads only operated across solution points reading the flux values from textured memory. The presented algorithm was found to be around 1% faster. The derivative of the flux is completed across the flux points and stored at the solution points (these coefficients are all in c_x and c_y). The final results are written to global memory for time-stepping.

3.2.5 CPR / NDG CUDA

Both CPR and NDG methods have the unique property that solution and flux point coincide with one another, which enables the entire algorithm to be written in one GPU kernel, as outlined in Figure 3.2. The major differences between the two methods are illustrated within the *CPR_Flux / NDG_Flux Part 1* and *CPR_Flux / NDG_Flux Part 2* kernels (Algorithms 10 and 11). If a comment line states *Only CPR*, then the lines which follow are only implemented with the CPR method (the same case for NDG). For CPR, the solution states are loaded into memory, the **for** loop computes the solution derivatives, and the projections are computed and stored. NDG requires the flux values to be stored and the flux derivatives computed within the **for** loop.

Algorithm 10 CPR_Flux / NDG_Flux Part 1

▷ Max of flux points or solution points
t_x = threadIdx.x
▷ Current element in the block and global element
t_y = threadIdx.y
k = blockIdx.x ∗ blockDim.y + t_y
▷ Solution points in x and y directions
$i_x = mod(t_x, n_{sp1d})$
$i_y = t_y / n_{sp1d}$
▷ Only CPR - Shared memory
__shared__ $q^s[size]$
▷ Only NDG - Shared memory
__shared__ $f^s[size]$
if $k <$ n$_e$ **then**
  ▷ Operate on solution points first
  if $t_x < n_{sp}$ **then**
    ▷ Only CPR - Load solution into shared memory from texture
    $q^s[(t_x, t_y, 0...n_v)] = q^t[(t_x, k, 0...n_v)]$
    ▷ Only NDG - Load solution from texture and store flux values
    $f^s[(t_x, t_y, 0...n_v)] = f(q^t[(t_x, k, 0...n_v)])$
    syncthreads()
    for $l = 0$ to n_{sp1d} **do**
      ▷ Only CPR
      $c_x = c_x^t[(l, i_x)]$
      $c_y = c_y^t[(l, i_y)]$
      $dq_x[(0...n_v)] = dq_x[(0...n_v)] + c_x * q^s([i_x, l, t_y, 0...n_v)]$
      $dq_y[(0...n_v)] = dq_y[(0...n_v)] + c_y * q^s([i_y, l, t_y, 0...n_v)]$
      ▷ Only NDG
      $d_x = d_x^t[(l, i_x)]$
      $d_y = d_y^t[(l, i_y)]$
      $df_x[(0...n_v)] = df_x[(0...n_v)] + d_x * f^s([i_x, l, t_y, 0...n_v)]$
      $df_y[(0...n_v)] = df_y[(0...n_v)] + d_y * f^s([i_y, l, t_y, 0...n_v)]$
    end for
    ▷ Only CPR - Compute projections
    $Proj[(0...n_v)] = P(dq_x, dq_y)$
  end if
  syncthreads()
  ...

Table 3.3: Shared memory requirements: *CPR_Flux* / *NDG_Flux*

P^k	n_{fp}	B_{base}	B_{SM}	Size	Bytes
P^1	8	64	8	256	2,048
-	-	128	16	512	4,096
-	-	256	32	1024	8,192
P^2	12	64	5	240	1,920
-	-	128	10	480	3,840
-	-	256	21	1008	8,064

From part 1 of the algorithm, the blocks and threads are defined, and shared memory is allocated. The threads run on either the number of solution points or flux points, whichever is greater. For P^1 reconstruction, $n_{sp} = 4$ while $n_{fp} = 8$ (two points per face), but for P^4 reconstruction, $n_{sp} = 25$ and $n_{fp} = 20$. In this thesis, only P^1, P^2, and P^3 reconstructions are considered, so threads will always run across flux points. The shared memory requirements for the CPR / NDG algorithm are outlined in Table 3.3. The byte count is similar to the *DG_Flux* requirements in Table 3.1, but slightly improved for P^2 reconstruction. Overall, the shared memory requirement is cheaper, since both DG and SD require two separate arrays of shared memory. CPR and NDG require only one shared memory array, a benefit of combining the solution and flux points. The 64 block base is used in this thesis for CPR/NDG.

The algorithm sets threads to operate over flux points or solution points, whichever is larger (the algorithm can then switch to operating on the other set within the kernel). First, operations are completed over solution points, reading in the solution states and storing the states (flux values for NDG) into shared memory. The shared memory is used in computing derivatives of the states in CPR, or the flux for NDG. The chain rule is used for the flux derivative in CPR and the projection term formulates the flux derivatives. Once the necessary derivatives are completed, the threads are halted using *syncthreads* to change the threads from operating on solution points to the flux points.

The second part of the algorithm switches the threads to operate on the total number of flux points, n_{fp}. The coupling of the elements is straightforward, as information is already on element interfaces. The same algorithm from *DG_Couple* (Algorithm 6) is used to compute the common Riemann flux. The flux is not written into global memory, but is rather stored into the register location f^n and used to

Algorithm 11 CPR_Flux / NDG_Flux Part 2

...
if $t_x < n_{fp}$ **then**
▷ Couple elements (see DG_Couple)
...
▷ Store normal flux difference into shared memory (f^s for NDG)
$q^s[(t_x, t_y, 0...n_v)] = f_x[(0...n_v)] * n_x + f_y[(0...n_v)] * n_y - f^n[(0...n_v)]$
end if
syncthreads()
if $t_x < n_{sp}$ **then**
▷ Get number of updates per solution point
$n_{upd} = n^t_{upd}[t_x]$
▷ Correct the normal flux (Lift the flux for NDG)
for $l = 0$ to n_{upd} **do**
▷ Locations for correction (lifting)
$id = id^t[(t_x, l, k])$
▷ Only CPR
$Corr[(0...n_v)] = Corr[(0...n_v)] - \alpha^t[id] * q^s[(t_x, t_y, 0...n_v)]$
▷ Only NDG
$Lift[(0...n_v)] = Lift[(0...n_v)] - L^t[id] * f^s[(t_x, t_y, 0...n_v)]$
end for
▷ Only CPR
$r^g[(t_x, k, 0...n_v)] = Proj[(0...n_v)] + Corr[(0...n_v)]$
▷ Only NDG
$r^g[(t_x, k, 0...n_v)] = df_x[(0...n_v)] + df_y[(0...n_v)] + Lift[(0...n_v)]$
end if
end if

compute the normal flux difference on the flux points. This difference is stored into the same shared memory location used in *CPR_Flux / NDG_Flux Part 1*, overwriting the existing memory. Once again, the threads are halted to ensure writing to the shared memory location is completed. The threads are switched again to operate on solution points. The normal flux difference is corrected (α coefficients), or lifted (L coefficients), at the interface and used to update the system. The main benefit of the CPR/NDG method is the ability to complete the spatial update stage in one kernel, with only one sweep through the domain. The other high-order methods require three separate kernels, corresponding to three different domain sweeps.

3.2.6 Shock Capturing

The shock capturing algorithm for FV differs significantly from the other high-order methods. The *FV_Reconstruct* algorithm can be modified to limit the slopes of the reconstructed solutions at the interfaces. The high-order schemes require new kernels to be written to appropriately handle the shock capturing algorithm. The FV shock capturing is shown in Algorithm 12, *FV_Reconstruct MUSCL*. The required solutions from neighboring elements are gathered and used with the *minmod* limiter to limit the solutions. The *minmod* limiter is defined as

Algorithm 12 FV_Reconstruct MUSCL

▷ Faces in element
t_x = threadIdx.x
▷ Current global face
k = blockIdx.x $*$ blockDim.x + threadIdx.y
if $k < \mathrm{n}_e$ **then**
  ▷ Gather information from neighbors
  $q_{e_1}[(0...n_v)] = q^t[id_{e_1}(t_x, k)]$
  $q_{e_2}[(0...n_v)] = q^t[id_{e_2}(t_x, k)]$
  ...
  ▷ Reconstruct left and right solutions with MUSCL
  $q_L[(0...n_v)] = q_L[(...n_v)] - \mathrm{minmod}(q_{e_1}, q_{e_2}, ...)$
  $q_R[(0...n_v)] = q_R[(...n_v)] + \mathrm{minmod}(q_{e_1}, q_{e_2}, ...)$
  ▷ Compute the interface flux
  InterfaceFlux(q_L, q_R, f_n)
  ▷ Store interface flux into global memory
  $f^g[k + (0...n_v) * n_f] = f_n[(0...n_v)]$
end if

$$\text{minmod}(a,b) = \begin{cases} a \text{ , if } |a| < |b| \text{ and } ab > 0 \\ b \text{ , if } |b| < |a| \text{ and } ab > 0 \\ 0 \text{ , if } ab \leq 0 \end{cases} \tag{3.2.1}$$

where a and b are two real inputs. A similar function can be written for three inputs as

$$\text{minmod}(a,b,c) = \begin{cases} a \text{ , if } |a| < |b| \text{ and} |a| < |c| \text{ and } abc > 0 \\ b \text{ , if } |b| < |a| \text{ and} |b| < |c| \text{ and } abc > 0 \\ c \text{ , if } |c| < |a| \text{ and} |c| < |b| \text{ and } abc > 0 \\ 0 \text{ , if } abc \leq 0 \end{cases} \tag{3.2.2}$$

where c is an additional real input. Depending on the order of the scheme, either a 2^{nd} or 3^{rd} or MUSCL reconstruction, Equations (3.2.1) and (3.2.2) are used respectfully.

The high-order methods require extra sweeps through the domain, since slope limiting requires solution averages which are not inherently calculated in a high-order method. The methods require first that solution averages be constructed in every element in the domain. Then the averages can be used to detect troubled elements and limit the solutions if needed. The kernel *Average*, shown in Algorithm

Algorithm 13 Average

▷ The current solution state
t_x = threadIdx.x
▷ The current element in the block
t_y = threadIdx.y
▷ Current global element
k = blockIdx.x ∗ blockDim.y + t_y
if $k < n_e$ **then**
  for $l = 0$ to n_{sp} **do**
    ▷ Build average
    $q_m[t_x] = c_m^t[l] * q^t[(t_x, l, k)]$
  end for
  $q_m^g[(t_x, k)] = q_m[t_x]$
end if

13, computes the average solution states within each element. The thread grid is $\vec{t} = \left[n_v, \frac{B_{base}}{n_v}\right]$ where $B_{base} = 64$. Each thread loops through the solution points within an element and computes the solution

average using average coefficients stored in c_m (computed using weights). The average is computed in registers and copied into the GPUs global memory, q_m^g.

Once the averages are constructed, the discontinuity detection can begin, completed in kernel *Limit Part 1* (Algorithm 14). The thread grid is $\vec{t} = \left[max(n_{sp}, n_{ep}), \frac{B_{base}}{max(n_{sp}, n_{ep})}\right]$, where n_{ep} is the number of edge points in an element (faces per element multiplied by points per face). A *max* function is needed to switch the threads from edge points to solution points. The solution is interpolated to the element edges if necessary (DG and SD). For NDG and CPR, the points already reside on element edges, and this step can be skipped. The indexes of neighboring elements is loaded to obtain the solution averages computed previously in q_m^g. The minmod limiter is used to construct a second solution at the edge, which is then compared to the interpolated solution. In the simulations presented, the value of $\epsilon = 1.0 \times 10^{-3}$. Each edge point now holds a 1 or 0, indicating troubled or not. This information must be broadcasted to the entire element (if one point is troubled, the element is troubled). The *Limit Part 2* kernel (Algorithm 15) completes this and applies slope limiting to the troubled elements.

Algorithm 14 Limit Part 1

▷ Solution points
t_x = threadIdx.x
▷ The current element in the block
t_y = threadIdx.y
▷ Current global element
k = blockIdx.x ∗ blockDim.y + t_y
if $k < n_e$ **then**
  ▷ Run through points on edges
  if $j < n_{ep}$ **then**
    ▷ Interpolate to edge if necessary
    q_l = ...
    ▷ Load index locations of neighboring elements
    (i_1, i_2) =...
    ▷ Construct the minmod at the edge
    $q_e = q_m^t[k] + minmod(q_m^t[k] - q_l, q_m^t[i_2] - q_m^t[k], q_m^t[k] - q_m^t[i_1])$
    ▷ Check if solution is large
    if $|q_l - q_e| > \epsilon$ **then**
      $mark = 1$
    end if
    ▷ Store the mark into shared memory at edge
    $tmp_s[(t_x, t_y)] = mark$
    syncthreads()
    ...

Algorithm 15 Limit Part 2

```
        ...
        mark = 0
        ▷ Each edge point runs through the others edge points
        for l = 0 to n_ep do
            mark = mark + tmp_s[(l, t_y)]
        end for
        mark_s[t_y] = mark
    end if
    syncthreads()
    if j < n_sp then
        if mark_s[t_y] > 0 then
            ▷ Apply slope limiting now
            ...
        end if
    end if
end if
```

Every edge point needs to see the markings of the others, which is completed using a summation. This way, an element with at least one troubled point will give each edge point a value of one. The marking is sent into shared memory so the information can be communicated when the threads switch to operate across the solution points. At each solution point, in shared block t_y, the marking is read from the shared space, and slope limiting is applied if this marking is greater than zero.

CHAPTER 4. GPU COMPARISON RESULTS

In this chapter several results from the GPU implementation are presented and discussed. The Finite Volume (FV), Discontinuous Galerkin (DG), Nodal Discontinuous Galerkin (NDG), Correction Procedure via Reconstruction (CPR), and Spectral Difference (SD) methods each simulate a smooth and discontinuous problem, and the computational work and solution errors are compared for varying grids and orders of accuracy.

4.1 Simulation Preliminaries

Before presenting the results, a discussion of the CFL condition, code compilation, and timings is needed. The CFL number for high-order methods is known to be quite restrictive in comparison to FV. To ensure a fair comparison with FV, the following convention is used: At the end of a simulation, the error is recorded. A new simulation is completed at a value of $0.5 * CFL$ of the previous. Again, the error is recorded. If the percent error between these two errors is less than 0.1%, the CFL is termed the maximum CFL. Errors were completed by comparing the averaged solution with the averaged exact solution. For P^2 FV, the error is computed by reconstructing the solution along element faces, and then using a quadrature rule to compute an averaged solution [1]. Finally, since FV has one solution state per element, while high-order methods have multiple, the total number of degrees of freedom (DOFs) between the methods is held constant. Only quadrilateral elements are considered in this work, hence for one element and a P^2 reconstruction with a high-order method, 9 DOFs occupy the element. To match this, the FV method must have 9 elements, since there is only one solution per element.

A single Tesla K20c GPU card is used for all simulations. The code is compiled under compute architecture 3.5 with the CUDA toolkit version 6.0. Compilation options include the `-O3` flag for maximum compiler optimizations. All codes are compiled for double precision computing and include

the CUDA 64-bit libraries. The computational time was nondimensionalized by taubench [121] using the following taubench condition: `./Taubench -n 250000 -s 10`. On the GPU workstation used in our simulations, taubench gave a value of 8.274. This produces what is known as a work unit, as suggested by the 1st International Workshop on High-Order Methods [1] when comparing timings from numerical methods across different computing platforms. A work unit is a nondimensionalized unit computed by dividing the computational time it takes to complete a simulation by the taubench result.

4.2 Smooth Problem

A vortex propagation case is the presented smooth problem in this thesis. The flow of the vortex is characterized by Shu [122]. A mean flow with density (ρ), velocities (u and v), and pressure (p) is specified $(\rho, u, v, p) = (1, 1, 0, 1)$ with fluctuation in the velocity, temperature (T), and entropy (S)

$$
\begin{aligned}
(\delta u, \delta v) &= \frac{\sigma}{2\pi} e^{0.5(1-r^2)} (-y, x) \\
\delta T &= -\frac{(\gamma - 1)\sigma^2}{8\gamma\pi^2} e^{1-r^2} \\
\delta S &= 0
\end{aligned}
$$

Here, $r^2 = x^2 + y^2$ and the vortex has strength $\sigma = 5$. An exact solution exists and can be found using $x_e = x - u_0 t$ and $y_e = y - v_0 t$, where t is the final time and u_0 and v_0 are the initial velocities. The solution evolves until time $t = 1$ and the L_2 error norm of ρ is computed. The domain is taken as $[-5, 5] \times [-5, 5]$ and periodic conditions are imposed on the boundaries. Discretizations from 20×20 to 100×100 quadrilateral elements are used in the high-order method simulations. Table 1 shows the maximum CFL chosen for the runs, which allowed a less than 0.1% error change when the CFL was decreased by $1/2$.

The solution errors versus work units for P^1, P^2, and P^3 reconstructions are shown in Figure 4.1. The overall trend of increasing accuracy with increasing work unit is observed for all methods. It is observed that the high-order methods obtain smaller error thresholds than the FV method for a given work unit (exception for P^2 NDG, which has larger errors associated). For P^1 reconstructions shown in Figure 4.1 (a), DG clearly outperforms other methods, but as the order is increased, Figure 4.1 (b) shows CPR and SD both achieve comparable errors with DG for a given work unit. In the case of CPR,

Table 4.1: Maximum CFL - Smooth problem

P^1—DOFs	CPR	NDG	SD	DG	FV
1600	0.24	0.24	0.3	0.24	0.4
3600	0.24	0.24	0.3	0.24	0.4
6400	0.24	0.24	0.3	0.24	0.38
10000	0.24	0.24	0.3	0.24	0.38
14400	0.24	0.24	0.3	0.24	0.37
P^2—DOFs	CPR	NDG	SD	DG	FV
3600	0.14	0.14	0.2	0.14	0.4
8100	0.13	0.13	0.2	0.13	0.4
14400	0.13	0.13	0.2	0.13	0.38
22500	0.13	0.13	0.2	0.13	0.37
32400	0.13	0.13	0.2	0.13	0.37

the schemes compact nature is attributed to this, where the operations to compute the flux derivative are contained in one GPU kernel. For the case of SD, Table 4.1 shows that the SD method can take larger time-steps than the other high-order methods, as the CFL is not as restrictive, which enables the method to arrive at solution errors for a comparable work unit. Similar plots are observed in the comparative study done by Yu *et al.* [27], where the error is compared with work units for several high-order methods on CPUs. They also observe that CPR has the lowest error given a work unit for P^2 reconstruction, while NDG is significantly higher. In the results presented here, SD is comparable to CPR because the maximum allowable time-step for a given temporal error is used. This is different than the approach by Yu *et al.* [27], where a constant time-step is implemented. A fourth order reconstruction is also completed, and shown in Figure 4.1 (c). It illustrates that the SD and CPR methods both obtain the lowest errors for a given work unit for a P^3 reconstruction.

Figure 4.2 shows the work unit needed to complete a simulation on a given mesh for P^1, P^2, and P^3 reconstructions. The obvious trend of the work unit increasing for finer meshes is observed for all orders of accuracy. To complete a full simulation, the FV and CPR methods are the fastest on coarse meshes (Figure 4.2 (a) and (b)). Small computational domains do not take advantage of the GPU architecture with the optimizations and different memory types discussed in this paper. As the domain is refined and the order is increased, the high-order schemes can produce solutions faster than FV. The data illustrates

that on fine meshes with high-order reconstruction, the CPR, NDG, and SD methods run faster than the FV method with increasing DOFs. Further increasing the order to P^3, Figure 4.2 (c), shows the CPR and NDG converge to the same work unit for a given simulation. The SD method, however, is able to complete solutions faster than any other high-order method.

The solution errors are recorded for the high-order methods and are shown in Tables 4.2, 4.3, and 4.4 for P^1, P^2, and P^3 reconstructions respectfully. The order of accuracy found from the error norms is computed as

$$m = \frac{\log(e_2/e_1)}{\log(g_2/g_1)} \tag{4.2.1}$$

where m is the slope, e_k is the error at point k, and g_k is defined as the inverse of the number of degrees of freedom. As an example, consider a 30×30 and 20×20 grid with P^1 reconstruction. The g values become

$$g_1 = \frac{1}{\sqrt{20 \times 20 \times 4}} = 0.025 \tag{4.2.2}$$

$$g_2 = \frac{1}{\sqrt{30 \times 30 \times 4}} = 0.167 \tag{4.2.3}$$

The slope for CPR is computed as

$$m = \frac{\log(1.46 \times 10^{-3}/3.21 \times 10^{-3})}{\log(0.167/0.025)} = 1.94 \tag{4.2.4}$$

The expected errors are observed in all three tables and are consistent with literature values [123, 124, 125, 19, 27]. For P^1 reconstructions in Table 4.2, all methods achieve the order of accuracy as indicated by the slope computations. In Table 4.3, a order of roughly 2.5 is observed between the high-order methods, with the exception of NDG, where aliasing errors are the cause for the drop in order. The P^3 results are shown in Table 4.4, where again the expected order of accuracy is achieved. The FV errors and order are shown in Table 4.5 for completeness. In order to compute the P^2 reconstruction error for FV, a sufficient quadrature approximation is used, as suggested by Wang *et al.* [1], where the reconstructed solution used for error calculations was the same as that used in the residual evaluation. This is only applicable for high-order FV, and the P^1 errors were computed simply using the solution averages.

Table 4.2: High-order error values for smooth problem (P^1 reconstruction).

Method	CPR Error	CPR Slope	NDG Error	NDG Slope
20×20	3.21E-003	-	2.70E-003	-
30×30	1.46E-003	1.94	1.10E-003	2.22
40×40	8.25E-004	1.98	6.04E-004	2.07
50×50	5.29E-004	1.99	3.83E-004	2.04
60×60	3.67E-004	2.00	2.65E-004	2.02
Method	SD Error	SD Slope	DG Error	DG Slope
20×20	3.07E-003	-	1.65E-003	-
30×30	1.38E-003	1.96	6.63E-004	2.24
40×40	7.78E-004	2.00	3.59E-004	2.13
50×50	4.96E-004	2.02	2.26E-004	2.08
60×60	3.44E-004	2.02	1.55E-004	2.06

Table 4.3: High-order error values for smooth problem (P^2 reconstruction).

Method	CPR Error	CPR Slope	NDG Error	NDG Slope
20×20	4.30E-004	-	5.63E-004	-
30×30	1.38E-004	2.81	2.30E-004	2.20
40×40	6.51E-005	2.60	1.26E-004	2.10
50×50	3.69E-005	2.55	7.81E-005	2.14
60×60	2.32E-005	2.55	5.23E-005	2.19
Method	SD Error	SD Slope	DG Error	DG Slope
20×20	4.00E-004	-	2.24E-004	-
30×30	1.36E-004	2.66	7.95E-005	2.55
40×40	6.40E-005	2.62	3.90E-005	2.48
50×50	3.57E-005	2.62	2.24E-005	2.48
60×60	2.21E-005	2.62	1.42E-005	2.50

Table 4.4: High-order error values for smooth problem (P^3 reconstruction).

Method	CPR Error	CPR Slope	NDG Error	NDG Slope
20×20	3.05E-005	-	9.05E-005	-
30×30	5.03E-006	4.45	1.50E-005	4.43
40×40	1.45E-006	4.32	4.47E-006	4.20
50×50	5.57E-007	4.29	1.77E-006	4.15
60×60	2.61E-007	4.16	8.67E-007	3.91
Method	SD Error	SD Slope	DG Error	DG Slope
20×20	3.58E-005	-	1.53E-005	-
30×30	6.25E-006	4.30	2.45E-006	4.51
40×40	1.83E-006	4.27	6.02E-007	4.88
50×50	7.11E-007	4.25	2.19E-007	4.53
60×60	3.31E-007	4.20	9.96E-008	4.32

Table 4.5: Finite volume error values for smooth problem.

Method	FV P^1 Error	FV P^1 Slope
40×40	4.53E-003	-
60×60	2.05E-003	1.95
80×80	1.15E-003	2.01
100×100	7.34E-004	2.02
120×120	5.08E-004	2.02
Method	FV P^2 Error	FV P^2 Slope
60×60	5.81E-004	-
90×90	2.09E-004	2.52
120×120	1.03E-004	2.45
150×150	6.09E-005	2.37
180×180	4.01E-005	2.30

Table 4.6: Maximum CFL for the discontinuous problem.

P^1—DOFs	CPR	NDG	SD	DG	FV
160k	0.2	0.2	0.3	0.22	0.58
640k	0.2	0.2	0.27	0.2	0.58
P^2—DOFs	CPR	NDG	SD	DG	FV
360k	0.1	0.1	0.18	0.08	0.54
1440k	0.1	0.1	0.18	0.08	0.54

4.3 Discontinuous Problem

The discontinuous problem is a radially expanding shock tube from Toro [118]. A domain of size $[-1, 1] \times [-1, 1]$ initializes density and pressure (ρ, p) of 1.0 inside a radius of 0.4. Outside the radius, $\rho = 0.125$ and $p = 0.1$. There is no velocity component at the initial time. Rather than using solution errors to check if the CFL is small enough, the residual error is used, as this problem has no analytic solution.

The solution is ran until a final time of $t = 0.25$, where the density is compared along the centerline, $y = 0$. For the reference solution, the data was taken from the text *Riemann Solvers and Numerical Methods for Fluid Dynamics* [118] (digitized for use here). The solution contours and density values are shown in Figure 4.3 (a) and (b). The density contours were generated using a P^1 CPR reconstruction on 160,000 elements. As mentioned previously, the density is gathered along the centerline and compared across all methods. As illustrated in Figure 4.3 (b), all methods have good agreement with the reference solution for P^1 reconstruction.

The total computational work per time-step is shown in Figure 4.4 for two computational grids. For the P^1 and P^2 reconstruction timings, the CPR method completes each iteration the cheapest, which is expected since all computations are completed within one GPU kernel. NDG follows a similar trend, with extra work required in the flux derivative calculations. SD and DG both require a significant amount of work per iteration when compared to the other methods. This is mostly attributed to the number of GPU kernels required per iteration. The FV method is comparable to the CPR method for small computational domains, but once the domain is large enough, the additional memory types used

for the CPR implementation become more important, and it is able to complete iterations faster than FV. While work per iteration is interesting, the total work to complete the solution is more important and is shown in Figures 4.5 and 4.6.

In Figure 4.5, a P^1 reconstruction is shown for two domains of 160,000, Figure 4.5 (a), and 640,000, Figure 4.5 (b), DOFs. The FV method is able to complete the solution faster than any other method shown, while the CPR method is the best performing high-order method for both grids. DG takes a significant amount of work to complete the simulation, due to the amount of kernels needed and the presence of surface and volume integrals in the formulation. For a P^2 reconstruction, a more interesting trend is observed. Once again, FV is the best performing method, but the SD method is only 5% slower, as shown in Figure 4.6 (a), and only 25% slower, as shown in Figure 4.6 (b)). This is due to the time-step the SD method can take computed from the CFL condition, which is 80% higher than the CPR method's CFL condition. The extra sweeps through the domain demanded by the high-order method algorithms, one to average the solution and another to apply limiting, is the reason the FV method is able to compute solutions faster, as the limiting within the FV algorithm is implemented in the flux computation, shown in Section 3.2.6.

4.4 Discussion of Results

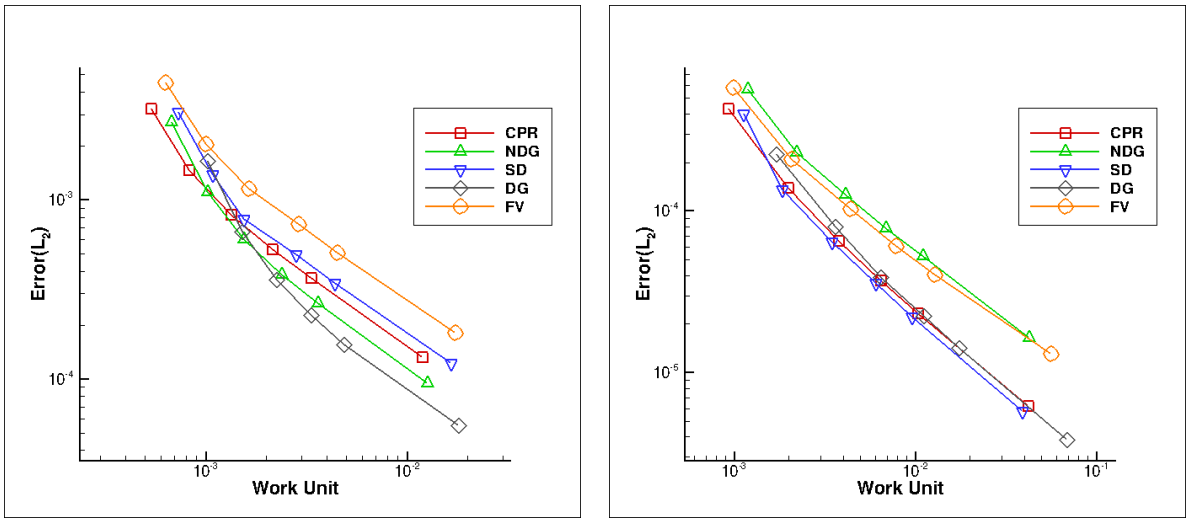
The presented results compared multiple numerical methods implemented using GPU CUDA computing, including FV, CPR, NDG, SD, and DG methods. A fair comparison was conducted for both smooth and discontinuous problems, where each method utilized their maximum allowable time-steps for stability. In addition, the number of DOFs were held constant for each grid and order of accuracy tested.

For smooth problems, it is apparent that high-order methods can not only achieve a given accuracy cheaper than FV, but they can also complete simulations faster. Results show that CPR, DG, and SD all perform significantly better than FV for both P^1 and P^2 reconstructions. NDG does not show comparable errors with the aforementioned methods, but does show comparable speeds with CPR. DG, while it yields low L_2 norm errors, takes longer to complete simulations. When increasing the order of accuracy, the SD method performs very well with the CPR method in terms of error norms. In addition,

SD shows the ability to complete solutions faster than any other method.

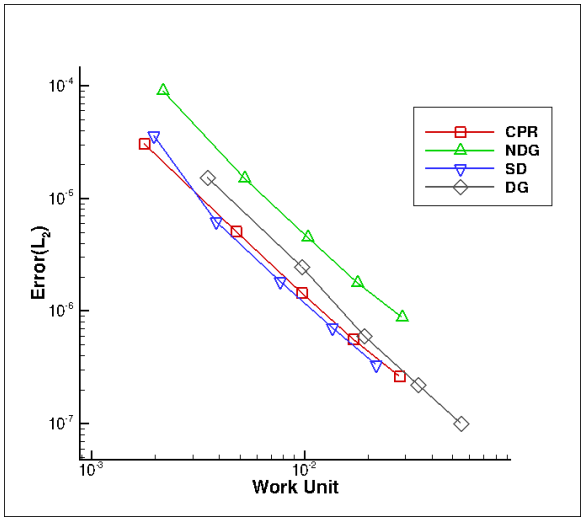
For discontinuous problems, all methods show good agreement with the solution. The key result is in the total work unit needed to complete a simulation. For low-order accurate reconstructions, results show that the FV method is the computationally cheapest method employed, while CPR is the cheapest high-order method. The DG method is the most expensive method, both per iteration and to complete a full simulation. For P^2 reconstructions, the FV and SD methods both perform comparably well, while the other methods are consistent with the P^1 reconstruction results.

The problems to be solved in this thesis contain solution discontinuities and will demand extremely small time-steps to simulate. It is the intent to use a high-order method for the simulations. From the results shown, not only does the SD method yield some of the lowest solution errors per work unit, it also is able to complete solutions faster when considering P^2 and P^3 reconstructions compared to other high-order methods. In addition, the SD method has a CFL condition which is not as restrictive as other high-order methods, which allows the method to take larger time-steps. This is very appealing for the problems desired to be simulated. It is due to these reasons that the SD method will be used for the remainder of this thesis.



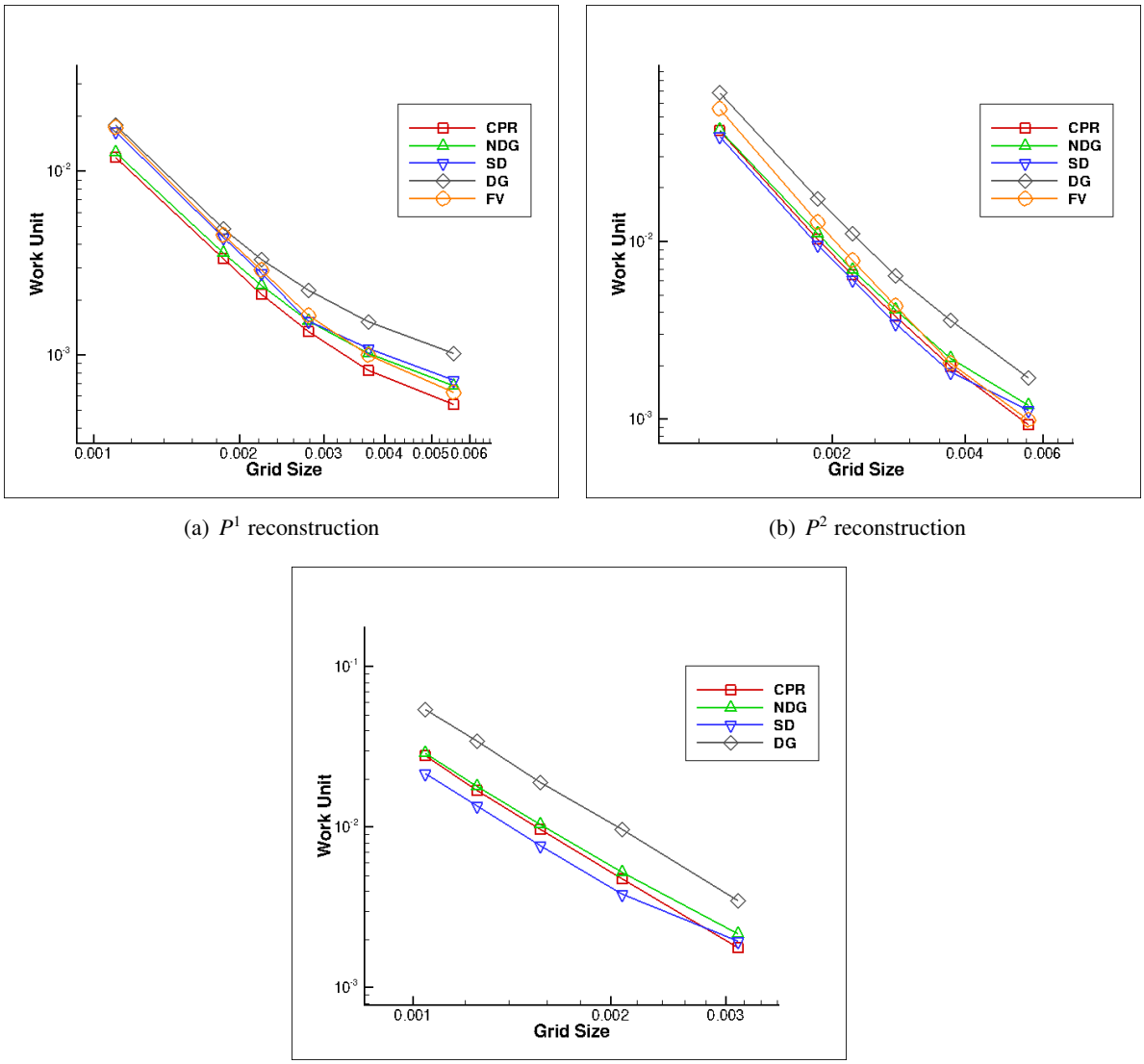
(a) P^1 reconstruction

(b) P^2 reconstruction



(c) P^3 reconstruction

Figure 4.1: Smooth problem L_2 density errors using (a) P^1, (b) P^2, and (c) P^3 reconstructions versus work unit.

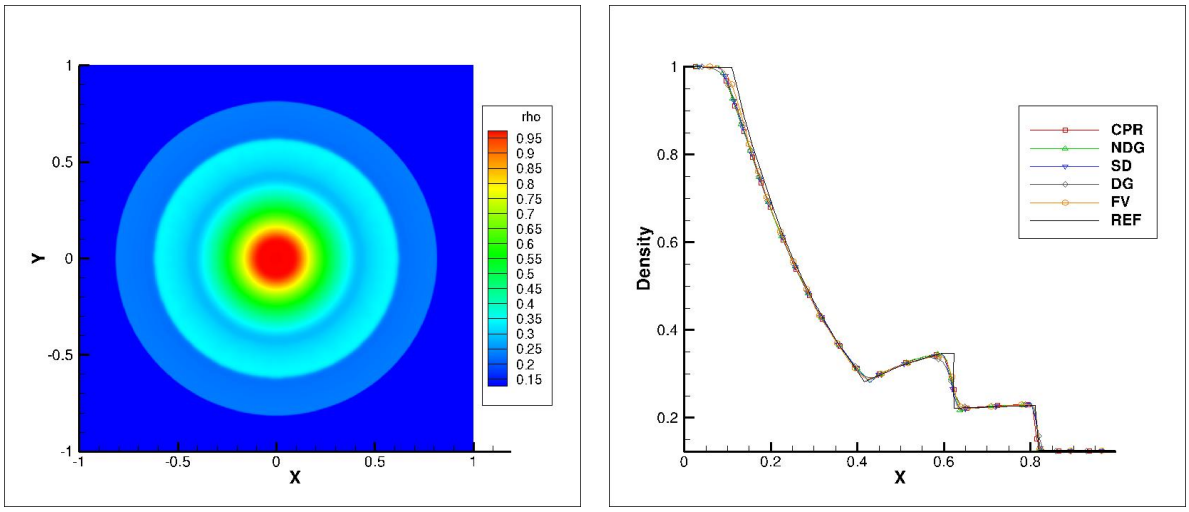


(a) P^1 reconstruction

(b) P^2 reconstruction

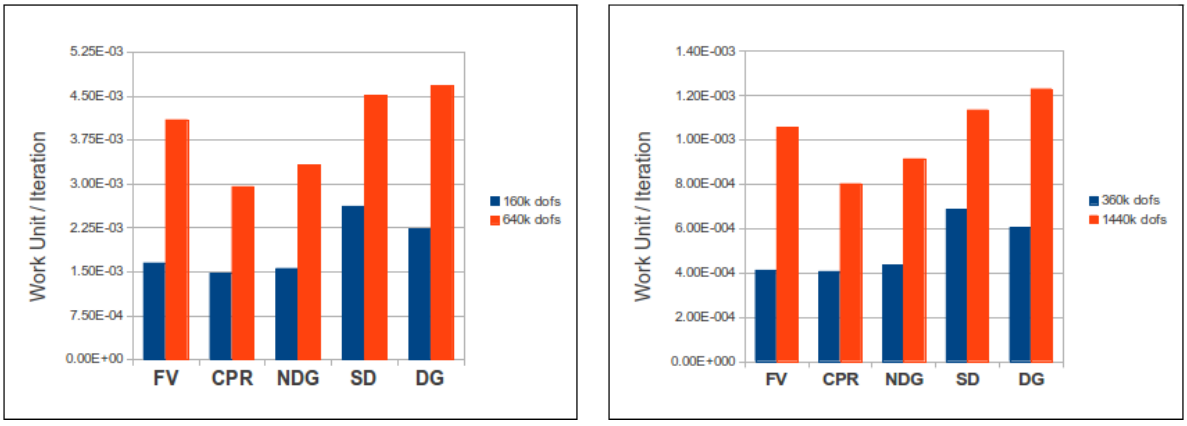
(c) P^3 reconstruction

Figure 4.2: Smooth problem total work unit to finish simulations for (a) P^1, (b) P^2, and (c) P^3 reconstructions.



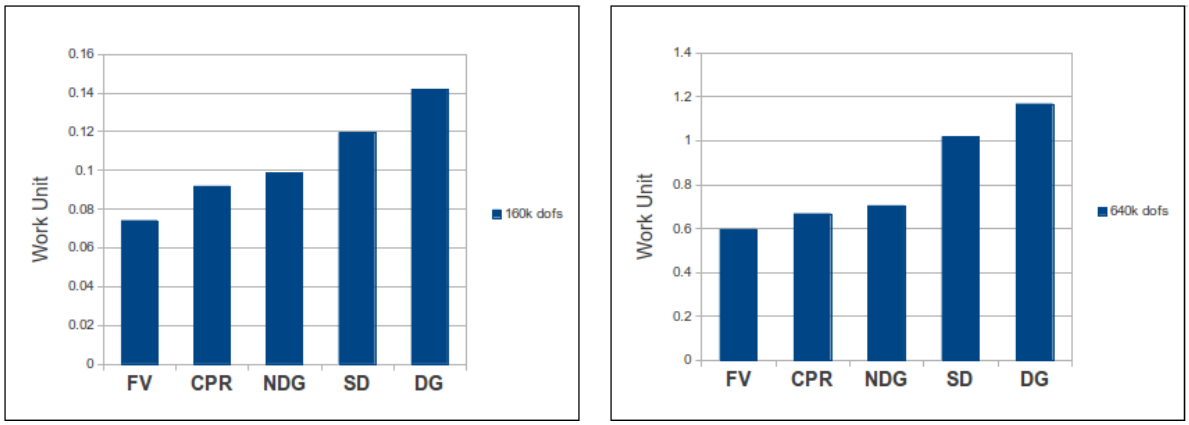
(a) CPR density contours (b) Method comparison

Figure 4.3: Discontinuous test case P^1 results (a) density contours (b) solution comparison along centerline.



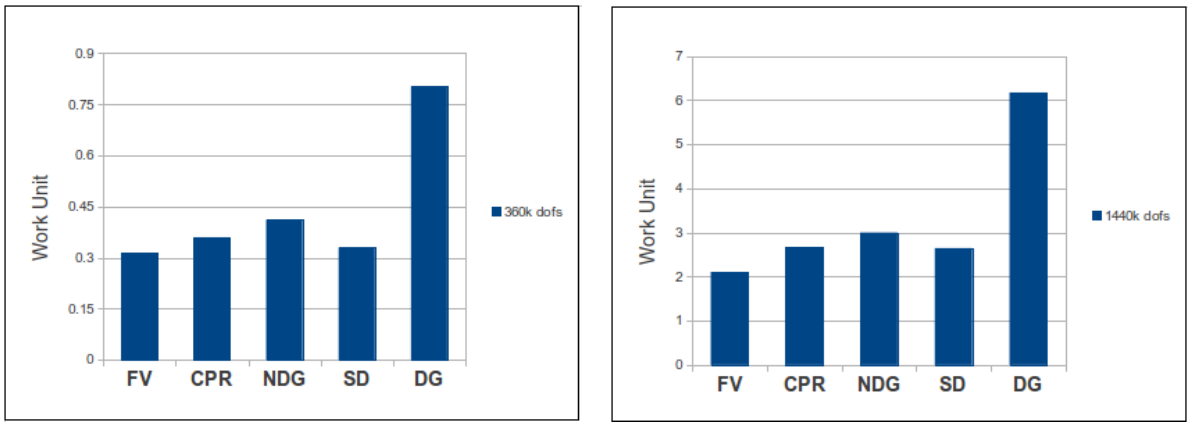
(a) P^1 reconstruction (b) P^2 reconstruction

Figure 4.4: Computational work per iteration for (a) P^1 and (b) P^2 reconstruction.



(a) (b)

Figure 4.5: Total work for P^1 reconstruction (a) 160,000 and (b) 640,000 DOFs.



(a) (b)

Figure 4.6: Total work for P^2 reconstruction (a) 360,000 and (b) 1,440,000 DOFs.

CHAPTER 5. MULTIPHASE NUMERICAL MODELING

The numerical simulation of multiphase flows has many important applications, including collapse of bubbles in flows, dynamics of fuel spray in engines, and hypervelocity impacts. This thesis is concerned with only hypervelocity impacts, but applications to other problems could be completed using the formulation presented. The governing equations are the two-dimensional Euler equations outlined in Chapter 2. While these equations capture the physics of single-component fluid problems, they do not properly define flows where the fluids posses differing equations of state, or even the same equation of state with differing parameters to define each fluid. This issue is addressed within this chapter, where the proper treatment of material interfaces is explored and the application of a multiphase model with the Spectral Difference (SD) method is outlined.

5.1 Multi-Component Flow Calculations

In order to model compressible inviscid fluids, the Euler equations are applied, which solve conservation of mass, momentum, and energy. While it is true that the conservative form best captures the flow description, this does not necessarily hold for multi-component flows. In fact, extending the Euler equations to handle multiple fluids in conservative form gives rise to pressure oscillations at the fluid interface. These oscillations are not small, and cannot be ignored as they give rise to density and velocity oscillations. One very interesting aspect is that these oscillations are apparent when a simple first order method is applied to the equations. This clears up any confusion that the oscillations are due to higher-order numerical schemes. In order to avoid these oscillations, it may be favorable to sacrifice the best flow description of conservative form and write the system in non-conservative form [89].

In order to numerically and mathematically explore the issues of multiphase modeling, three distinct models are implemented for the one-dimensional Euler system. The system follows the hyperbolic conservation law with a matrix in front of the flux derivative, written as

$$\frac{\partial \boldsymbol{q}}{\partial t} + \mathbf{A}\frac{\partial \boldsymbol{f}(\boldsymbol{q})}{\partial x} = 0 \tag{5.1.1}$$

where $\mathbf{A}$ is a coefficient matrix and the solution and flux vectors are written as the following:

$$\boldsymbol{q} = \begin{pmatrix} \rho \\ \rho u \\ e \end{pmatrix}, \qquad \boldsymbol{f}(\boldsymbol{q}) = \begin{pmatrix} \rho u \\ p + \rho u^2 \\ u(e+p) \end{pmatrix} \tag{5.1.2}$$

The pressure is still computed according to the ideal gas model, Equation (2.0.6), with $v = 0$.

5.1.1 Problem Setup

A one-dimensional problem will be investigated which involves the propagation of a material interface. Consider a domain $x \in [0, 1]$ which is discretized with 200 elements. The pressure and velocity are both unity everywhere ($p = 1$ and $u = 1$). There is a density discontinuity at $x = 0.2$ where $\rho_L = 1.0$ and $\rho_R = 0.1$. Since the pressure and velocity are both unity, this density front should advect to the right at a speed of $u = 1$ and at a constant pressure. The issue arises when the following condition is imposed: Assume the left and right fluids have a differing ratio of specific heats, where $\gamma_L = 1.4$ and $\gamma_R = 1.2$. The front should still travel from left to right at the same speed, but results will show that in some models, this is not the case.

The three models will be labeled as Model I, Model II, and Model III. The conservation law needs to be augmented to support the differing γ values in the problem. The Model I formulation inserts the γ term directly into conservation form, and is written as the following:

$$\boldsymbol{q} = \begin{pmatrix} \rho \\ \rho u \\ e \\ \rho\gamma \end{pmatrix}, \qquad \boldsymbol{f}(\boldsymbol{q}) = \begin{pmatrix} \rho u \\ p + \rho u^2 \\ u(e+p) \\ \rho\gamma u \end{pmatrix}, \qquad \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.1.3}$$

One additional equation is added to the system to track the interface γ value. This is a fully conservative model. Model II is a quasi-conservative model, written as the following:

$$\boldsymbol{q} = \begin{pmatrix} \rho \\ \rho u \\ e \\ \gamma \end{pmatrix}, \qquad \boldsymbol{f}(\boldsymbol{q}) = \begin{pmatrix} \rho u \\ p + \rho u^2 \\ u(e+p) \\ \gamma \end{pmatrix}, \qquad \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & u \end{bmatrix} \tag{5.1.4}$$

The added equation to track γ is written in primitive form with the matrix outside the flux vector. This model is very similar to Model III, with a small change to the γ variable in the equation. Model III is written as the following:

$$\boldsymbol{q} = \begin{pmatrix} \rho \\ \rho u \\ e \\ \frac{1}{\gamma-1} \end{pmatrix}, \qquad \boldsymbol{f}(\boldsymbol{q}) = \begin{pmatrix} \rho u \\ p + \rho u^2 \\ u(e+p) \\ \frac{1}{\gamma-1} \end{pmatrix}, \qquad \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & u \end{bmatrix} \tag{5.1.5}$$

where the γ variable is changed to $1/(\gamma - 1)$. The reason for this change will be made clear in the next section.

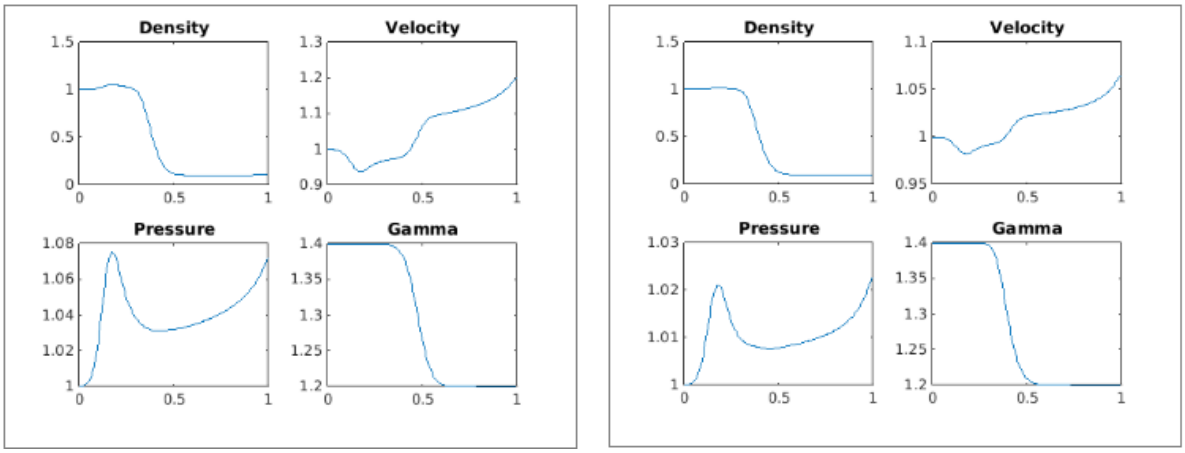
The equations are solved using a first order upwinding scheme. This scheme is written as the following:

$$\boldsymbol{q}_j^{n+1} = \boldsymbol{q}_j^n - \frac{\Delta t}{\Delta x}\mathbf{A}\left[\boldsymbol{f}_{j+1/2} - \boldsymbol{f}_{j-1/2}\right] \tag{5.1.6}$$

where n is the time level, j is the grid point, Δt is the time-step, Δx is the grid spacing, and $\boldsymbol{f}_{j+1/2}$ is the numerical flux evaluated using the Rusanov flux discussed in Section 2.2.3.1. A constant time step of $\Delta t = 0.0005$ was used with a single stage Runge-Kutta scheme.

5.1.2 Numerical Results

The numerical simulation results illustrate some very interesting features. All models were simulated to a final time of $t = 0.2$ and are shown in Figure 5.1. Figure 5.1 (a) shows the fully conservative results from Model I. At the very first time step, a pressure spike is observed at the material front. The spike grows with time, propagates through the domain, and leads to changes in the velocity and density fields. Notice how the pressure changes have affected the entire domain, before the density front should



(a) Model I

(b) Model II

(c) Model III

Figure 5.1: Multicomponent flow simulation with three methods

arrive. These unphysical results are generated from only the first order numerical scheme solving the conservative equations. Even using the primitive form for γ in Model II shows material jumps, shown in Figure 5.1 (b). However, it should be noted that the pressure spike is much less severe, and the resulting changes in density and velocity are reduced from the Model I results. In both models, the advection of γ shows no oscillations, but Model I does advect the γ field farther than it should have physically traveled. Model II does improve on this feature.

The Model III results in Figure 5.1 (c) are physically accurate. The material front and γ field have advected at the same speed, and no spikes in pressure or velocity fields are observed. These results beg the question, why does the conservative approach produce such spikes in the results? Also, why does

changing the primitive formulation to $1/(\gamma - 1)$ remove all non-physical results? Both these questions are explored in the next section.

5.1.3 Numerical Interface Analysis

Consider a left and right state in a computational domain, where there is only a material interface and no density gradient. This condition is written as

$$\rho_L = \rho_R = \rho$$

$$u_L = u_R = u$$

$$p_L = p_R = p$$

$$\gamma_L \neq \gamma_R$$

Consider Model I (fully conservative equations) for analysis. From the first order upwind update, the conservation of mass is written as

$$\rho_j^{n+1} = \rho_j^n - \lambda \left[(\rho u)_{j+1/2}^n - (\rho u)_{j-1/2}^n \right] \tag{5.1.7}$$

where $\lambda = \Delta t / \Delta x$. From the Rusanov solver, the right interface flux is written as

$$(\rho u)_{j+1/2}^n = \frac{1}{2} \left[(\rho u)_j^n + (\rho u)_{j+1}^n - |v_n| \left(\rho_{j+1}^n - \rho_j^n \right) \right] \tag{5.1.8}$$

where $|v_n|$ is the maximum wave speed at the interface. Since the density and velocities are equal throughout the domain, the flux becomes the following

$$(\rho u)_{j+1/2}^n = (\rho u)^n \tag{5.1.9}$$

A similar result is found using the left interface, which becomes

$$(\rho u)_{j-1/2}^n = (\rho u)^n \tag{5.1.10}$$

Thus, the conservation of mass simply becomes the following

$$\rho_j^{n+1} = \rho_j^n \tag{5.1.11}$$

The conservation of momentum follows a similar analysis, where the update relation is written as the following

$$(\rho u)_j^{n+1} = (\rho u)_j^n - \lambda \left[\left(p + \rho u^2 \right)_{j+1/2}^n - \left(p + \rho u^2 \right)_{j-1/2}^n \right] \tag{5.1.12}$$

If the flux terms are expanded using the Rusanov flux (Section 2.2.3.1), the following right interface flux is obtained as

$$\left(p+\rho u^2\right)^n_{j+1/2} = \frac{1}{2}\left[p^n_j + (\rho u^2)^n_j + p^n_{j+1} + (\rho u^2)^n_{j+1} - |v_n|\left((\rho u)^n_{j+1} - (\rho u)^n_j\right)\right] \tag{5.1.13}$$

Again, since the density, velocity, and pressure profiles are constant throughout the domain, the flux becomes the following

$$\left(p+\rho u^2\right)^n_{j+1/2} = \frac{1}{2}\left[p^n_j + (\rho u^2)^n_j + p^n_j + (\rho u^2)^n_j\right] \tag{5.1.14}$$

The left flux follows a similar analysis, and the resulting update is written as

$$(\rho u)^{n+1}_j = (\rho u)^n_j \tag{5.1.15}$$

The conservation of energy becomes slightly more involved. The energy update takes the form

$$e^{n+1}_j = e^n_j - \lambda\left[(u(e+p))^n_{j+1/2} - (u(e+p))^n_{j-1/2}\right] \tag{5.1.16}$$

In order to complete the analysis, the energy must be related to the pressure

$$e = \frac{p}{\gamma - 1} + \frac{1}{2}\rho u^2 \tag{5.1.17}$$

In this relation, note that p, ρ, and u are all constant through the domain, but γ defines the material front and is not constant. For simplicity, let's assume that $\gamma_{j-1} \neq \gamma_j = \gamma_{j+1}$, which indicates the material interface is located at the left edge of the cell. With this assumption, the right energy flux becomes the following

$$\begin{aligned}(u(e+p))^n_{j+1/2} = \frac{1}{2}\left[u^n_{j+1}\left(\frac{p^n_{j+1}}{\gamma^n_{j+1}-1} + \frac{1}{2}(\rho u^2)^n_{j+1} + p^n_{j+1}\right) + u^n_j\left(\frac{p^n_j}{\gamma^n_j - 1} + \frac{1}{2}(\rho u^2)^n_j + p^n_j\right)\right] \\ -\frac{1}{2}\left[|v_n|\left(\frac{p^n_{j+1}}{\gamma^n_{j+1}-1} + \frac{1}{2}(\rho u^2)^n_{j+1} - \frac{p^n_j}{\gamma^n_j - 1} - \frac{1}{2}(\rho u^2)^n_j\right)\right]\end{aligned} \tag{5.1.18}$$

Since every variable on indexes j and $j+1$ are equivalent, the formulation reduces to the following

$$(u(e+p))^n_{j+1/2} = \frac{1}{2}\left[u^n_j\left(\frac{p^n_j}{\gamma^n_j - 1} + \frac{1}{2}(\rho u^2)^n_j + p^n_j\right) + u^n_j\left(\frac{p^n_j}{\gamma^n_j - 1} + \frac{1}{2}(\rho u^2)^n_j + p^n_j\right)\right] \tag{5.1.19}$$

The left interface is written as

$$\begin{aligned}(u(e+p))^n_{j-1/2} = \frac{1}{2}\left[u^n_j\left(\frac{p^n_j}{\gamma^n_j - 1} + \frac{1}{2}(\rho u^2)^n_j + p^n_j\right) + u^n_{j-1}\left(\frac{p^n_{j-1}}{\gamma^n_{j-1} - 1} + \frac{1}{2}(\rho u^2)^n_{j-1} + p^n_{j-1}\right)\right] \\ -\frac{1}{2}\left[|v_n|\left(\frac{p^n_j}{\gamma^n_j - 1} + \frac{1}{2}(\rho u^2)^n_j - \frac{p^n_{j-1}}{\gamma^n_{j-1} - 1} - \frac{1}{2}(\rho u^2)^n_{j-1}\right)\right]\end{aligned} \tag{5.1.20}$$

Since the density and velocity are constant in the flow, several terms can be canceled. The flux becomes the following

$$(u(e+p))^n_{j-1/2} = \frac{1}{2}\left[u^n_j\left(\frac{p^n_j}{\gamma^n_j - 1} + \frac{1}{2}(\rho u^2)^n_j + p^n_j\right) + u^n_{j-1}\left(\frac{p^n_{j-1}}{\gamma^n_{j-1} - 1} + \frac{1}{2}(\rho u^2)^n_{j-1} + p^n_{j-1}\right)\right] - \frac{1}{2}\left[|v_n|\left(\frac{p^n_j}{\gamma^n_j - 1} - \frac{p^n_{j-1}}{\gamma^n_{j-1} - 1}\right)\right] \tag{5.1.21}$$

The energy update now takes the following form:

$$e^{n+1}_j = e^n_j - \frac{\lambda}{2}\left[u^n p^n\left(\frac{1}{\gamma^n_j - 1} - \frac{1}{\gamma^n_{j-1} - 1}\right) + |v_n| p^n\left(\frac{1}{\gamma^n_j - 1} - \frac{1}{\gamma^n_{j-1} - 1}\right)\right] \tag{5.1.22}$$

The final update relation is for the γ advection. The update is computed by

$$(\rho\gamma)^{n+1}_j = (\rho\gamma)^n_j - \lambda\left[(\rho\gamma u)^n_{j+1/2} - (\rho\gamma u)^n_{j-1/2}\right] \tag{5.1.23}$$

The Rusanov flux allows the update to be written as

$$(\rho\gamma)^{n+1}_j = (\rho\gamma)^n_j - \frac{\lambda}{2}\left[\rho u\left(\gamma^n_j - \gamma^n_{j-1}\right) - |v_n|\rho\left(\gamma^n_j - \gamma^n_{j-1}\right)\right] \tag{5.1.24}$$

Since the density is constant for all time from Equation (5.1.11), the γ update relation becomes

$$\gamma^{n+1}_j = \gamma^n_j - \frac{\lambda}{2}\left[u\left(\gamma^n_j - \gamma^n_{j-1}\right) - |v_n|\left(\gamma^n_j - \gamma^n_{j-1}\right)\right] \tag{5.1.25}$$

All the update relations are presented here for clarity:

$$\rho^{n+1}_j = \rho^n_j \tag{5.1.26}$$

$$(\rho u)^{n+1}_j = (\rho u)^n_j \tag{5.1.27}$$

$$e^{n+1}_j = e^n_j - \lambda\left[u^n p^n\left(\frac{1}{\gamma^n_j - 1} - \frac{1}{\gamma^n_{j-1} - 1}\right) + |v_n| p^n\left(\frac{1}{\gamma^n_j - 1} - \frac{1}{\gamma^n_{j-1} - 1}\right)\right] \tag{5.1.28}$$

$$\gamma^{n+1}_j = \gamma^n_j - \lambda\left[u\left(\gamma^n_j - \gamma^n_{j-1}\right) - |v_n|\left(\gamma^n_j - \gamma^n_{j-1}\right)\right] \tag{5.1.29}$$

where $\lambda = \lambda/2$ for simplicity. To show how the model generates non-physical results, consider the pressure computation after the first time-step. The pressure is computed using

$$p^{n+1}_j = \left(\gamma^{n+1}_j - 1\right)\left[e^{n+1}_j - \frac{1}{2}\left(\rho u^2\right)^{n+1}_j\right] \tag{5.1.30}$$

With the update relations presented in Equations (5.1.26) - (5.1.29), the pressure at the next time level is written as the following

$$p_j^{n+1} = \left[\gamma_j^n - \lambda\left(u\Delta\gamma^n + |v_n|\Delta\gamma^n\right) - 1\right]\left\{e_j^n - \lambda\left[up\left(\frac{1}{\gamma_j^n - 1} - \frac{1}{\gamma_{j-1}^n - 1}\right) + |v_n|p\left(\frac{1}{\gamma_j^n - 1} - \frac{1}{\gamma_{j-1}^n - 1}\right)\right] - \frac{1}{2}\rho u^2\right\} \tag{5.1.31}$$

where $\Delta\gamma = \gamma_j - \gamma_{j-1}$. The relation can be rearranged in the following form

$$\begin{aligned} p_j^{n+1} &= \left(\gamma_j^n - 1\right)\left(e_j^n - \frac{1}{2}\rho u^2\right) \\ &-\lambda\left(\gamma_j^n - 1\right)\left[up\left(\frac{1}{\gamma_j^n - 1} - \frac{1}{\gamma_{j-1}^n - 1}\right) + |v_n|p\left(\frac{1}{\gamma_j^n - 1} - \frac{1}{\gamma_{j-1}^n - 1}\right)\right] \\ &-\lambda\left(u\Delta\gamma^n + |v_n|\Delta\gamma^n\right)\left\{e_j^n - \lambda\left[up\left(\frac{1}{\gamma_j^n - 1} - \frac{1}{\gamma_{j-1}^n - 1}\right) + |v_n|p\left(\frac{1}{\gamma_j^n - 1} - \frac{1}{\gamma_{j-1}^n - 1}\right)\right] - \frac{1}{2}\rho u^2\right\} \end{aligned} \tag{5.1.32}$$

where the first term in the equation is just the pressure at time level n. Hence the pressure at the next time level takes the form

$$p_j^{n+1} = p_n^n + \delta p \tag{5.1.33}$$

where δp is a pressure spike which occurs near the material fronts where $\gamma_j \neq \gamma_{j-1}$. The incorrect update of the energy relation yields non-physical jumps in the pressure. While this analysis was completed for Model I, a similar analysis can be investigated for Model II. However, from the numerical results, it is clear that this model also generates non-physical results. In order to correctly update the pressure, changes in the model must occur.

It was observed in numerical results that Model III correctly computed the pressure and material fronts. To understand why the model yields correct solutions, a necessary condition is imposed, that $p_j^{n+1} = p_j^n$. To impose such a condition, consider Equation (5.1.17) for two time steps:

$$e_j^{n+1} = \frac{p_j^{n+1}}{\gamma_j^{n+1} - 1} + \frac{1}{2}\rho u^2 \tag{5.1.34}$$

$$e_j^n = \frac{p_j^n}{\gamma_j^n - 1} + \frac{1}{2}\rho u^2 \tag{5.1.35}$$

Method Recall that the density and velocity profiles are constant for all time. By imposing the pressure equality, the following can be written:

$$(\gamma_j^{n+1} - 1)\left[e_j^{n+1} - \frac{1}{2}\rho u^2\right] = (\gamma_j^n - 1)\left[e_j^n - \frac{1}{2}\rho u^2\right] \tag{5.1.36}$$

For convenience, we will set $\kappa = (\gamma - 1)$ and divide the κ terms to obtain

$$\frac{1}{\kappa_j^n}\left[e_j^{n+1} - \frac{1}{2}\rho u^2\right] = \frac{1}{\kappa_j^{n+1}}\left[e_j^n - \frac{1}{2}\rho u^2\right] \tag{5.1.37}$$

The e_j^{n+1} update has already been computed from the previous analysis in Equation (5.1.28). Inserting the energy update into the above equation yields the following

$$\frac{1}{\kappa_j^{n+1}}\left[e_j^n - \frac{1}{2}\rho u^2\right] = \frac{1}{\kappa_j^n}\left\{e_j^n - \lambda\left[u^n p^n\left(\frac{1}{\kappa_j^n} - \frac{1}{\kappa_{j-1}^n}\right) + |v_n| p^n\left(\frac{1}{\kappa_j^n} - \frac{1}{\kappa_{j-1}^n}\right)\right] - \frac{1}{2}\rho u^2\right\} \tag{5.1.38}$$

Once again, the definition of energy is used in the above to yield the following

$$\begin{aligned}\frac{1}{\kappa_j^{n+1}}\left[\frac{p_j^n}{\kappa_j^n} + \frac{1}{2}\rho u^2 - \frac{1}{2}\rho u^2\right] = \\ \frac{1}{\kappa_j^n}\left\{\frac{p_j^n}{\kappa_j^n} + \frac{1}{2}\rho u^2 - \lambda\left[u^n p^n\left(\frac{1}{\kappa_j^n} - \frac{1}{\kappa_{j-1}^n}\right) + |v_n| p^n\left(\frac{1}{\kappa_j^n} - \frac{1}{\kappa_{j-1}^n}\right)\right] - \frac{1}{2}\rho u^2\right\}\end{aligned} \tag{5.1.39}$$

Several terms cancel, and recall that $p_j = p$. The above equation can be written as

$$\frac{1}{\kappa_j^{n+1}}\left[\frac{p^n}{\kappa_j^n}\right] = \frac{1}{\kappa_j^n}\left\{\frac{p^n}{\kappa_j^n} - \lambda\left[u^n p^n\left(\frac{1}{\kappa_j^n} - \frac{1}{\kappa_{j-1}^n}\right) + |v_n| p^n\left(\frac{1}{\kappa_j^n} - \frac{1}{\kappa_{j-1}^n}\right)\right]\right\} \tag{5.1.40}$$

Factoring out the pressure and κ terms gives the following

$$\frac{1}{\kappa_j^{n+1}} = \frac{1}{\kappa_j^n} - \lambda\left[u^n\left(\frac{1}{\kappa_j^n} - \frac{1}{\kappa_{j-1}^n}\right) + |v_n|\left(\frac{1}{\kappa_j^n} - \frac{1}{\kappa_{j-1}^n}\right)\right] \tag{5.1.41}$$

where $\lambda = \lambda/2$. Note that this equation is simply the discretization of the following:

$$\frac{\partial}{\partial t}\left(\frac{1}{\kappa}\right) + u\frac{\partial}{\partial x}\left(\frac{1}{\kappa}\right) = 0 \tag{5.1.42}$$

with $\kappa_j = \kappa_{j+1}$. Hence, the analysis shows that discretizing the Euler system with the above added non-conservative equation will produce physically accurate results for material fronts with varying γ values. This formulation is completed for only one additional varying term. Additional equations would be required for more complex equations of state, where each varying term across the material front would need to be discretized in a manner similar to Equation (5.1.42). To avoid such complexities, a different model is considered for implementation.

5.2 Diffused-Interface Method

In order to model the transition regions between materials without generating non-physical pressure perturbations, the Diffused-Interface Method (DIM) is adapted. The model uses volume fractions to determine material characteristics based on equations of state which define each material. The Euler system is modified to allow volume fractions to be modeled. The numerical model and integration into the SD method are covered in this section.

5.2.1 Numerical Model

From the results presented in the previous section, an additional equation is needed to track the material interface, which can be completed by discretizing $1/\kappa$ in Equation (5.1.42). The added equation in the DIM, however, tracks the volume fractions. For a one-dimensional problem with two fluid phases, where each fluid is governed by a similar equation of state, the system becomes the following:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u)}{\partial x} = 0 \tag{5.2.1}$$

$$\frac{\partial \rho u}{\partial t} + \frac{\partial \left(\rho u^2 + p\right)}{\partial x} = 0 \tag{5.2.2}$$

$$\frac{\partial e}{\partial t} + \frac{\partial \left[u(e+p)\right]}{\partial x} = 0 \tag{5.2.3}$$

$$\frac{\partial \alpha_1}{\partial t} + u\frac{\partial \alpha_1}{\partial x} = 0 \tag{5.2.4}$$

where α_1 defines the phase, and the second fluid phase is $\alpha_2 = 1 - \alpha_1$. When more than two fluid phases are considered, this system cannot be used. This case will be investigated later. The one-dimensional model is now a 4-equation system with the added volume fraction equation. It is more convenient to cast the volume fraction equation into conservative form with an added source term. The spatial derivative can be written as

$$u\frac{\partial \alpha_1}{\partial x} = \frac{\partial (u\alpha_1)}{\partial x} - \alpha_1 \frac{\partial u}{\partial x} \tag{5.2.5}$$

The system can then be written in hyperbolic form with an added source term for the DIM formulation. The following system is obtained

$$\frac{\partial \boldsymbol{q}}{\partial t} + \frac{\partial \boldsymbol{f}}{\partial x} = \boldsymbol{S} \tag{5.2.6}$$

where the vectors are written as follows:

$$\boldsymbol{q} = \begin{pmatrix} \rho \\ \rho u \\ e \\ \alpha_1 \end{pmatrix}, \qquad \boldsymbol{f} = \begin{pmatrix} \rho u \\ p + \rho u^2 \\ u(e+p) \\ u\alpha_1 \end{pmatrix}, \qquad \boldsymbol{S} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \alpha_1 \frac{\partial u}{\partial x} \end{pmatrix} \tag{5.2.7}$$

The model now has an appropriate equation to track the material phase fronts throughout the simulation. In general, more than two phases are present in simulations, and multiple dimensional cases are of interest.

For a general two-dimensional problem, one may have n different phases to model. The same hyperbolic conservation law can be applied, with an added flux term for the second dimension. The governing equation is written as

$$\frac{\partial \boldsymbol{q}}{\partial t} + \frac{\partial \boldsymbol{f}}{\partial x} + \frac{\partial \boldsymbol{g}}{\partial y} = \boldsymbol{S} \tag{5.2.8}$$

Assume that there exists three or more material phases to be modeled ($n > 2$). In this example, each materials density and phase must both be tracked. The vectors in Equation (5.2.8) are written as follows:

$$\boldsymbol{q} = \begin{pmatrix} \rho\alpha_k \\ \rho u \\ \rho v \\ e \\ \alpha_k \end{pmatrix}, \qquad \boldsymbol{f} = \begin{pmatrix} \rho u \alpha_k \\ p + \rho u^2 \\ \rho uv \\ u(e+p) \\ u\alpha_k \end{pmatrix}, \qquad \boldsymbol{g} = \begin{pmatrix} \rho v \alpha_k \\ \rho uv \\ p + \rho v^2 \\ v(e+p) \\ v\alpha_k \end{pmatrix}, \qquad \boldsymbol{S} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \alpha_k \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) \end{pmatrix} \tag{5.2.9}$$

where α_k is the volume fraction of phase k. This yields a system of $(2n + 3)$ equations to be solved at every point in the domain. The total density is just the summation of each phases density contribution, written as

$$\rho = \sum_{i=1}^{k} \rho\alpha_k \tag{5.2.10}$$

From the density, the remaining primitive variables, velocities and pressures, can be computed as normal.

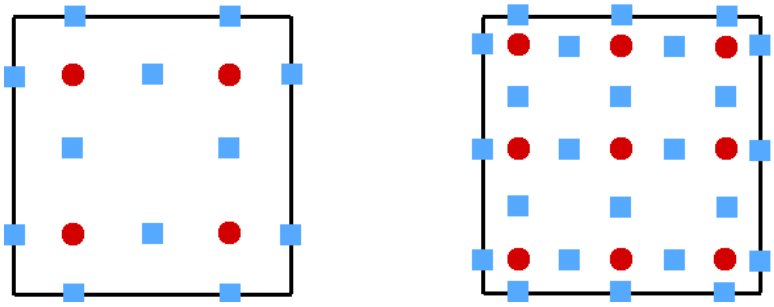


Figure 5.2: Spectral Difference elements with solution points (red circles) and flux points (blue squares) for P^1 and P^2 reconstructions

5.2.2 Implementation with SD

Implementation of the model into SD (which will be called SD-DIM) is rather straightforward. The overall SD algorithm can be broken down into three major parts:

1. Given the states at the solution points, interpolate the information to the flux points. Use the interpolated values to compute the flux vectors at the flux points.

2. Given the states at the flux points, couple the elements together using a Riemann solver at each interface to compute a Riemann flux. Store the flux at the interface flux points.

3. Given the flux values at the flux points, compute the flux derivatives and interpolate the derivatives to the solution points to update the solution states.

For clarity, P^1 and P^2 reconstructions are shown in Figure 5.2 to differentiate between solution and flux points. The above algorithm changes slightly when using the DIM. In step (1) and (2), the calculated pressures need to be augmented to support different equations of state, which will be discussed in Section 5.3. In step (3), additional derivatives are required for the source vector addition. The u and v velocities can be computed at the flux points from the interpolated solution state vector. The velocity derivatives are then computed using the following:

$$\left(\frac{\partial u}{\partial \xi}\right)_{i,j} = \sum_{r=0}^{k+1} u_{r+1/2,j} l'_{r+1/2}(\xi_i) \tag{5.2.11}$$

$$\left(\frac{\partial v}{\partial \eta}\right)_{i,j} = \sum_{r=0}^{k+1} v_{i,r+1/2} l'_{r+1/2}(\eta_j) \tag{5.2.12}$$

This is extremely convenient, as the $l'(\xi_i)$ and $l'(\eta_i)$ coefficients have already been computed to complete the flux derivatives (see Table 2.3). No extra memory storage is required, with the exception of storing the additional states for the partial differential equations. This makes implementation of SD-DIM extremely cost efficient.

5.3 Equations of State

In the GPU comparison results presented in Chapter 4, only the ideal gas equation of state was considered. For the problems considered, this state equation does not properly handle the physics. Two different equations of state are considered for implementation, the stiffened gas and Mie-Grüneisen state equations.

5.3.1 Stiffened Gas

When materials are under high pressures it is possible to describe them physically using the stiffened gas equation of state [109, 126, 127]. Under this assumption, the stiffened gas equation of state will be used for most simulations presented in this thesis. The internal energy is computed as

$$e_i = \frac{p + \gamma p_\infty}{\gamma - 1} \tag{5.3.1}$$

where γ is the ratio of specific heats and p_∞ is a pressure coefficient dependent on material properties. From Equation (5.3.1), setting the p_∞ term to zero yields the ideal gas equation of state. It follows that the pressure is computed as

$$p = (e_i - \gamma p_\infty)(\gamma - 1) \tag{5.3.2}$$

Each material is allowed to have differing γ and p_∞ terms, hence the volume fractions are needed in computing the pressure. To stay consistent with the model (see Section 5.1.3), the terms must be computed in the following manner:

$$\frac{1}{\gamma - 1} = \sum_k \frac{\alpha_k}{\gamma_k - 1} \tag{5.3.3}$$

$$\gamma p_\infty = \frac{\sum_k \frac{\alpha_k \gamma_k p_{o,k}}{\gamma_k - 1}}{\sum_k \frac{\alpha_k}{\gamma_k - 1}} \tag{5.3.4}$$

This will guarantee no pressure oscillations at the material interface. Finally, the speed of sound is computed as

$$c = \sqrt{\frac{\gamma(p + p_\infty)}{\rho}} \tag{5.3.5}$$

5.3.2 Mie-Grüneisen

The Mie-Grüneisen equation of state [128] is more complicated than the stiffened gas model, and it depends on each materials densities. While it is not used for the asteroid disruption problems, it is implemented to show the ability to utilize more complex equations of state. The pressure is described by

$$p = \left(e_i + \frac{p_{ref}}{\Gamma} - \rho e_{ref}\right) / \left(\frac{1}{\Gamma}\right) \tag{5.3.6}$$

where Γ, p_{ref}, and e_{ref} are all functions of density. The Grüneisen coefficient, Γ, takes the form

$$\Gamma = \Gamma_0 \left(\frac{V}{V_0}\right)^\alpha \tag{5.3.7}$$

where $\Gamma_0 = 1 - \gamma_0$ represents the Grüneisen coefficient at the initial density, $V = \frac{1}{\rho}$, and $\alpha = 1$ in the current work. The reference values for pressure and energy are

$$p_{ref} = p_0 + \frac{c_0^2(V_0 - V)}{[V_0 - s(V_0 - V)]^2} \tag{5.3.8}$$

$$e_{ref} = e_0 + \frac{1}{2}(p_{ref} + p_0)(V_0 - V) \tag{5.3.9}$$

where p_0, e_0, c_0, and s all depend on the material to be modeled. In a similar manner to the stiffened equation of state, the volume fractions can be applied to compute material mixtures throughout the domain. The speed of sound for the Mie-Grüneisen equation of state is

$$c^2 = \left[\Gamma + 1 + \frac{\rho}{\Gamma}\frac{d\Gamma}{d\rho}\right]\left(\frac{p - p_{ref}}{\rho}\right) + \frac{\Gamma p_{ref}}{\rho} + \frac{dp_{ref}}{d\rho} - \Gamma\rho\frac{de_{ref}}{d\rho} \tag{5.3.10}$$

The derivatives can be computed analytically and are shown here for completeness:

$$\frac{d\Gamma}{d\rho} = -\Gamma\frac{\rho_0}{\rho^2} \tag{5.3.11}$$

$$\frac{dp_{ref}}{d\rho} = \frac{2c^2 s\left(\frac{1}{\rho} + \frac{1}{\rho_0}\right)}{\rho^2\left[s\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right) + \frac{1}{\rho_0}\right]^3} - \frac{c^2}{\rho^2\left[s\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right) + \frac{1}{\rho_0}\right]^2} \tag{5.3.12}$$

$$\frac{de_{ref}}{d\rho} = \frac{c^2\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)}{2\rho^2\left[s\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right) + \frac{1}{\rho_0}\right]^2} + \frac{c^2\left(\frac{1}{\rho} + \frac{1}{\rho_0}\right)}{2\rho^2\left[s\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right) + \frac{1}{\rho_0}\right]^2} - \frac{c^2 s\left(\frac{1}{\rho} + \frac{1}{\rho_0}\right)\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)}{\rho^2\left[s\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right) + \frac{1}{\rho_0}\right]^2} \tag{5.3.13}$$

5.4 Multiphase Damage Modeling

To simulate damage or fracturing of the asteroid targets, a method to detect damaged points in the domain is necessary. It is important to note that our model is two-dimensional, hence area computations are required instead of volume computations. Each element has a specified initial density associated with it due to starting conditions. This density will be known as ρ_0. Since density can be related to volume (area), the ratio of the initial density inside an element relative to the new density can be written as

$$\frac{\rho_0}{\rho} = \frac{V}{V_0} \tag{5.4.1}$$

Assume that each element edge has an initial length of l_0, and when a new density occupies the element, the edges are changed by some amount Δ. This action is shown in Figure 5.3, where an initial element is expanded to encompass a new density value. Equation (5.4.1) can be written in terms of the element edges as

$$\frac{V}{V_0} = \frac{(l_0 + \Delta)^2}{l_0^2} = \left(\frac{l_0 + \Delta}{l_0}\right)^2 = \left(\frac{l_0}{l_0} + \frac{\Delta}{l_0}\right)^2 \tag{5.4.2}$$

In Equation (5.4.2), the volume terms are expanded to give the ratio of the change in length over the initial length. This ratio is defined as the strain the element experiences, ϵ. Hence, Equation (5.4.1) can be written as

$$\frac{\rho_0}{\rho} = (1 + \epsilon)^2 \tag{5.4.3}$$

which can easily be solved for the strain as

$$\epsilon = \sqrt{\frac{\rho_0}{\rho}} - 1 \tag{5.4.4}$$

If $\rho_0 > \rho$, then the strain will be positive, and material is in tension, while if $\rho_0 < \rho$, the strain will be negative which indicates the material is compressed. The final step is to compare the computed strain from the model with the maximum compressive, ϵ_c, or maximum tensile, ϵ_t, strain the material can experience. In the report by Stowe [129], several granite specimens are tested for maximum compressive and tensile strengths. The average of the tests are recorded within the report. From the averaged results, a maximum compressive strain of $\epsilon_c \approx 2000.0\mu$ strain and a maximum tensile stain of $\epsilon_t \approx 200.0\mu$ strain are the thresholds for damage detection.

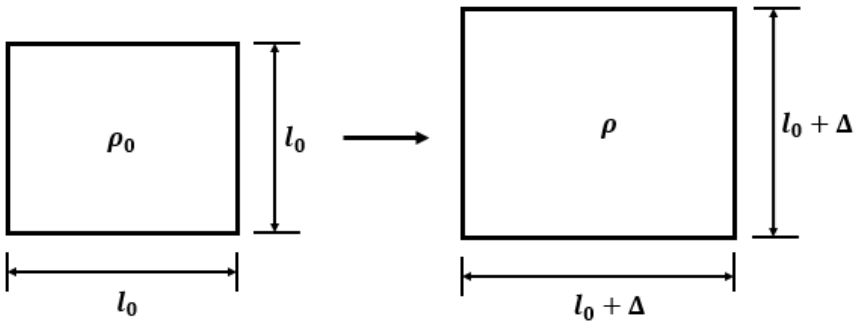


Figure 5.3: Damage characterization in one element

Once the strain computed from Equation (5.4.4) exceeds the maximum allowable strain in compression or tension, the region is marked as damaged. Of course, if the target is marked as damaged and immediately allowed to break apart, then the damaging shock wave will not travel through the target. An example blast wave is illustrated in Figure 5.4 [88], where if the target fragments at the peak value of the positive part (also called positive phase) the remainder of the positive part may not travel. At this peak point, the material is compressed, and waves should be allowed to propagate through this compressed region. A unique property of blast waves and impacts is the appearance of negative parts, which occur when the shock front has traveled a certain distance and the pressure behind the front drops below that of the surrounding atmosphere or material [73, 88, 130]. Once this wave part arrives, the damaging wave front no longer effects the region. The negative part arrival of the wave dictates when the material will experience damage. The issue now is how to numerically damage the point in the domain, such that fracture of the material occurs. To accomplish this, a change of material phase is completed. The basic concept is the following:

1. Compute the strain at every point and check to see if damage occurs. If it does, proceed.
2. Check if the negative part of the wave has arrived. If it has, proceed.
3. Complete a material phase change where the damaged material looses a percentage of its stiffness.
4. Update the new phases in $\boldsymbol{q}$ and continue to the next time-step.

The issue now is what material phase change in step (3) is appropriate for the given problem. Further explanation and analysis is discussed in Section 7.1.3.

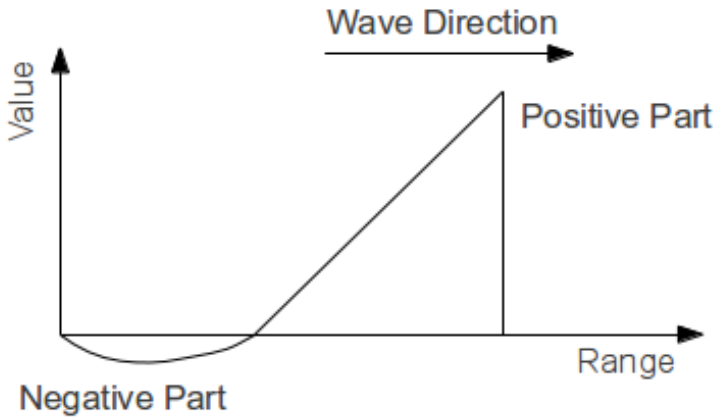


Figure 5.4: Illustration of a standard blast wave.

CHAPTER 6. NUMERICAL VERIFICATION CASES

Several verification cases are presented for the developed computer code. All cases utilize Graphics Processing Units (GPUs) with the Spectral Difference (SD) method for numerical simulations. As discussed in Section 2.2.4, a three-stage Runge-Kutta scheme is applied for time-marching with a constant time-step defined for each test case.

6.1 Sod's Shock Tube

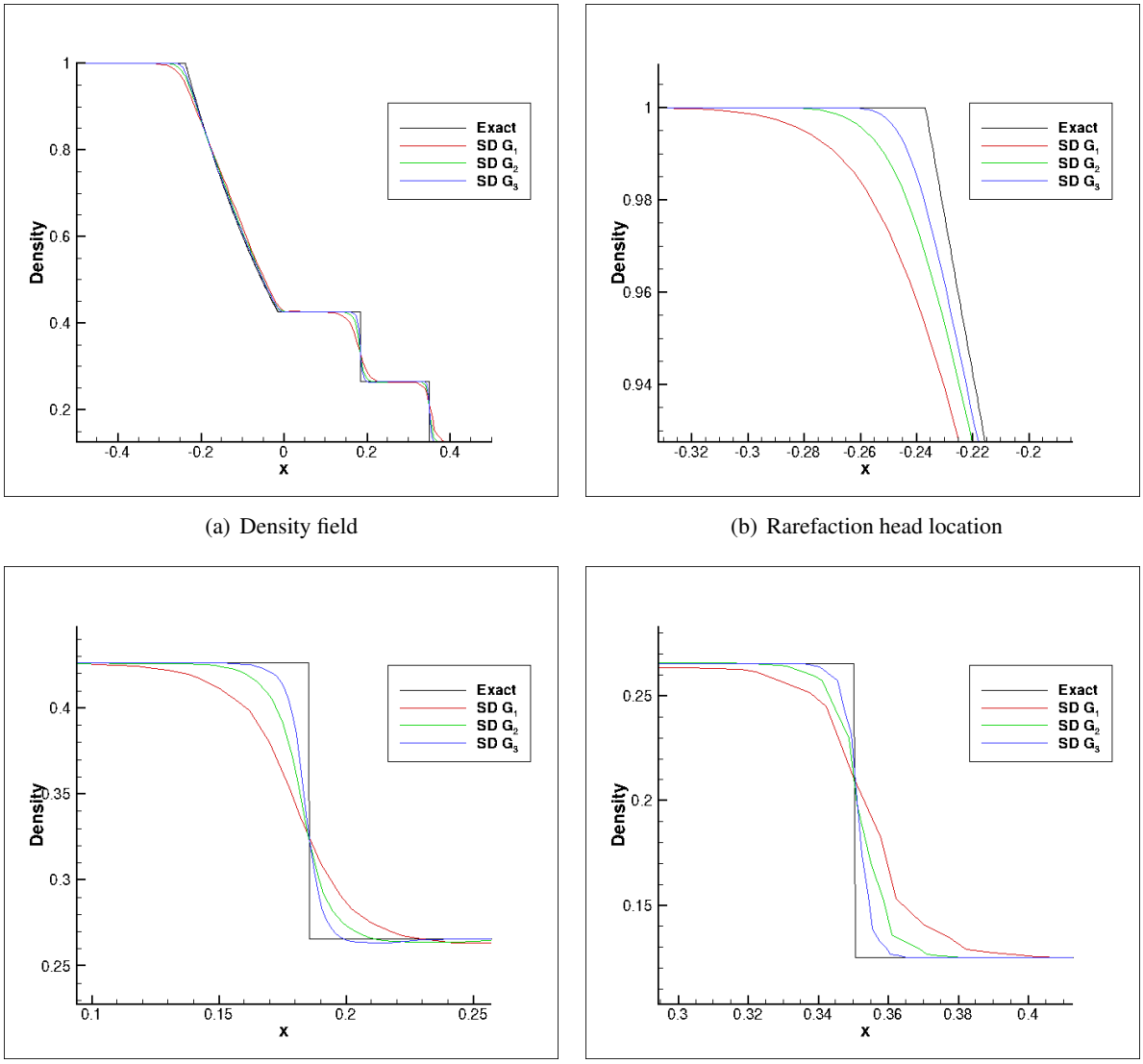
The standard Sod Shock Tube problem [131] was simulated to show proper implementation of the SD method with the limiter. The initial conditions are as follows:

$$(\rho, u, p)_L = (1.0, 0.0, 1.0), \quad (\rho, u, p)_R = (0.125, 0.0, 0.1)$$

where the left and right separation location is $x = 0$ in a domain from $x \in [-0.5, 0.5]$. The ideal gas equation of state is used with a ratio of specific heats, $\gamma = 1.4$. The Rusanov Riemann solver is used at the interfaces for element coupling (see Section 2.2.3.1). Three different computational grids are investigated, where 50, 100, and 200 elements discretized the grids G_1, G_2, and G_3 respectfully. A constant time-step of $\Delta t = 2.0 \times 10^{-4}$ is used to simulate until the computational time is $t = 0.2$ seconds. The solution reconstruction was held at P^2 for the grid refinement study.

Figure 6.1 (a) shows the density profiles, where three major regions of interest are observed. The rarefaction wave ($x \approx -0.3 - 0$) region, the contact discontinuity ($x \approx -0.2$), and the shock front ($x \approx -0.35$). The rarefaction head is blown-up in Figure 6.1 (b), where it is observed that grid refinement leads to closer agreement with the exact solution. A similar trend is also observed in Figure 6.1 (c) and in Figure 6.1 (d) at the contact discontinuity and shock front locations.

A second set of simulations are completed where the grid is held constant at G_1, while the order of accuracy is increased from P^1 to P^3, shown in Figure 6.2. It is observed that the solutions are similar



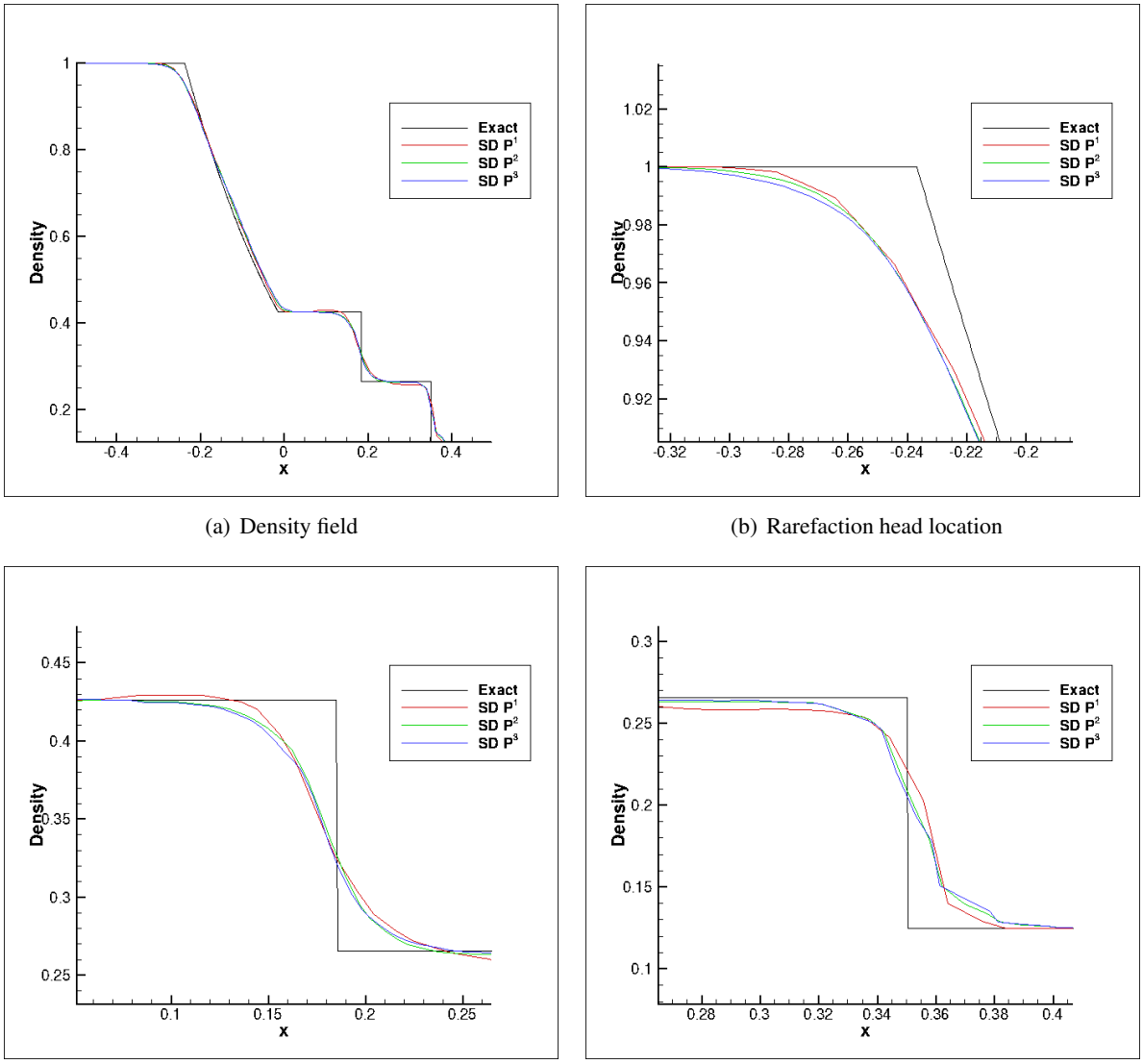
(a) Density field

(b) Rarefaction head location

(c) Contact discontinuity location

(d) Shock front location

Figure 6.1: Sod shock tube grid refinement (t = 0.2 seconds).



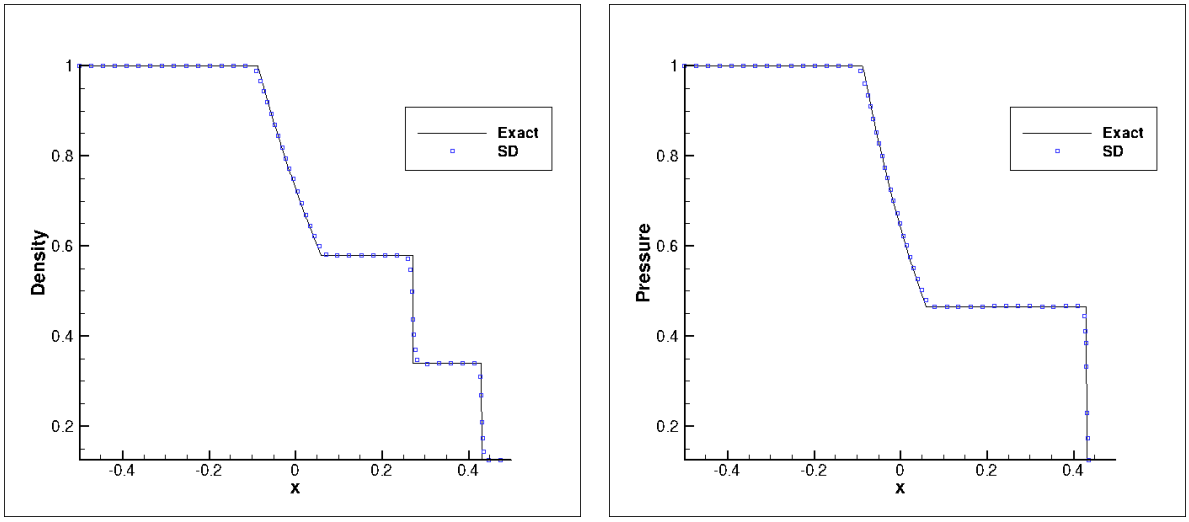
(a) Density field

(b) Rarefaction head location

(c) Contact discontinuity location

(d) Shock front location

Figure 6.2: Sod shock tube order refinement (t = 0.2 seconds).



(a) Density field (b) Pressure field

Figure 6.3: Strong shock problem ($t = 0.05$ seconds).

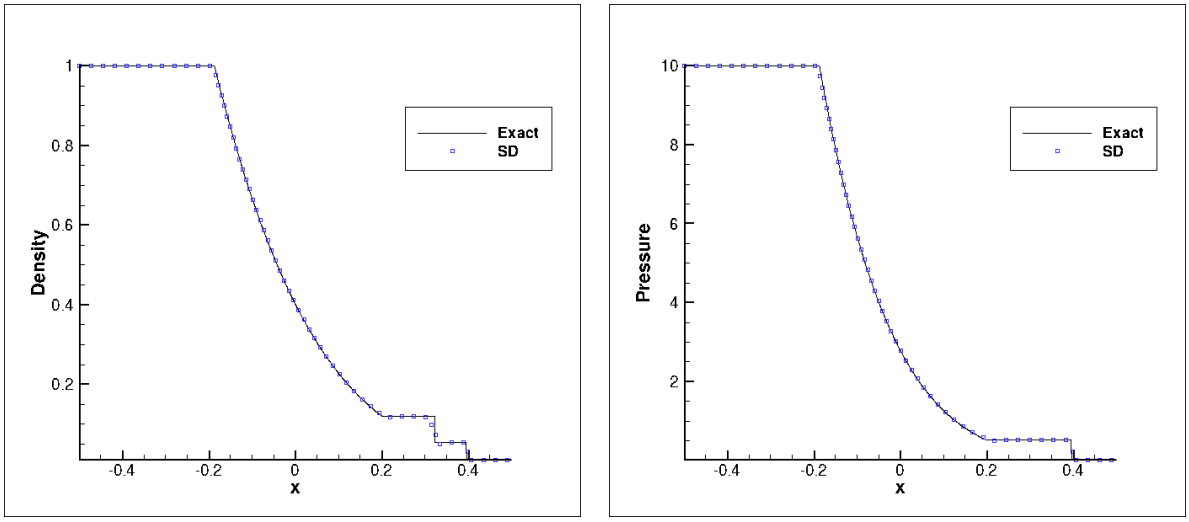
in all three regions. One major difference lies in the reconstruction in the smooth regions, between the rarefaction wave and the contact discontinuity and between the contact discontinuity and the shock wave. Figure 6.2 (c) and (d) show that increasing the order of accuracy yields more accurate solutions within the constant density regions, while the solutions around the discontinuities is quite similar.

6.2 Modified Sod's Shock Tube

The next test case initializes a moving discontinuity which is known to produce entropy violations by generating an expansion shock in the rarefaction region. The domain, $x \in [-0.5, 0.5]$, is discretized with 200 elements and a P^2 reconstruction with the Rusanov interface solver applied for coupling. The interface is located at $x = 0$, with left and right initial conditions as follows:

$$(\rho, u, p)_L = (1.0, 0.75, 1.0), \quad (\rho, u, p)_R = (0.125, 0.0, 0.1)$$

The ideal gas equation of state is used with $\gamma = 1.4$. The results for density and pressure are shown in Figure 6.3 (a) and (b). The numerical solution is shown with symbols, where each symbol is equally spaced at a 1% distance. From the density profile, Figure 6.3 (a), no expansion shock is created within the rarefaction region, and the numerical solution compares well with the exact solution.



(a) Density field (b) Pressure field

Figure 6.4: Strong shock problem ($t = 0.05$ seconds).

6.3 Strong Shock

While the Sod's shock tube case discussed in Section 6.1 demonstrates the capability of the solver to capture some basic features, it does not illustrate means of dealing with problems which contain multiple scales. This test case will show the solvers ability to handle such situations. A strong shock problem is simulated, where a narrow density peak is formed just behind the shock front. The domain, $x \in [-0.5, 0.5]$, is discretized with 400 elements and a P^2 reconstruction with the Rusanov interface solver. The initial conditions are as follows:

$$(\rho, u, p)_L = (1.0, 0.0, 10.0), \quad (\rho, u, p)_R = (0.01, 0.0, 0.01)$$

where the separation point between the states is again at $x = 0$. Again, the ideal gas equation of state is applied with $\gamma = 1.4$. These conditions generate an initial pressure ratio of 1000 at the interface. The solution is shown at a final time of $t = 0.05$ with a time-step of $\Delta t = 1.25 \times 10^{-5}$. The solutions for the density and pressure fields are shown in Figure 6.4 (a) and (b). The results indicate that the solver does possess the ability to capture problems with large density and pressure ratios, a capability desired in order to solve the intended problems.

6.4 Two-Fluid Shock Tube

This test case is a multifluid version of Sod's shock tube problem discussed in Section 6.1 [132]. The initial conditions are identical to Sod's problem, but the specific heat ratio is allowed to vary at the interface. The conditions are as follows:

$$(\rho, u, p, \alpha_1)_L = (1.0, 0.0, 1.0, 1.0), \quad (\rho, u, p, \alpha_1)_R = (0.125, 0.0, 0.1, 0.0)$$

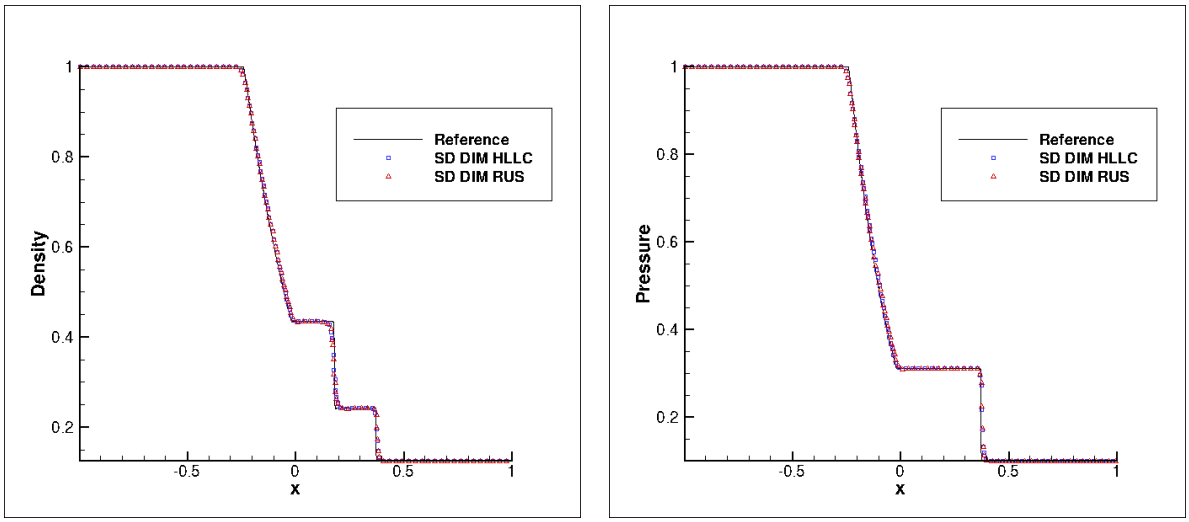
where the two fluids are now governed by the stiffened equation of state, discussed in Section 5.3.1, with $p_\infty = 0$, $\gamma_L = 1.4$, and $\gamma_R = 1.6$. The domain, $x \in [-1.1]$, is discretized with 200 elements and a P^2 reconstruction, and the density, pressure, and γ profiles are shown at a final time of $t = 0.2$ with a time step of $\Delta t = 2.0 \times 10^{-4}$. The solution is shown in Figure 6.5 and is compared with a reference solution, generated by running the SD method with the HLLC interface flux (Section 2.2.3.2) over 2000 elements. The Diffused-Interface Model (DIM) is needed for this problem, as the γ variation at the interface indicates two fluids with different equation of state properties. Figures 6.5 (a) - (c) show the density, pressure, and γ fields for two interface solvers, the Rusanov and HLLC Riemann solvers. The results indicate that both interface solvers capture the reference solution well, and no pressure oscillations are generated by the numerical model. One must now question the usage of the HLLC Riemann solver when a Rusanov flux appears sufficient. The next case illustrates how the Rusanov flux can still produce pressure oscillations even with SD-DIM.

6.5 Propagating Material Front

Consider the following test case in a domain $x \in [-1, 1]$. At $x = 0$, a material interface is defined with the following initial conditions:

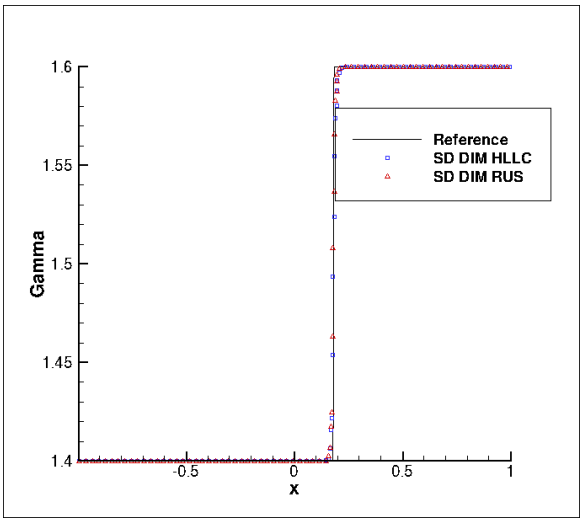
$$(\rho, u, p, \alpha_1)_L = (1.0, 1.0, 1.0, 1.0), \quad (\rho, u, p, \alpha_1)_R = (0.125, 1.0, 1.0, 0.0)$$

with $\gamma_L = 1.4$ and $\gamma_R = 1.6$. This produces a propagating material front moving from left to right at a speed of unity. Since the pressure and velocities are constant, the density and γ discontinuities should simply travel to the right, with no oscillations. The simulation is carried out until 0.1 seconds with $\Delta t = 2.0 \times 10^{-4}$. A P^1 reconstruction is applied to show high-order reconstructions are not the cause



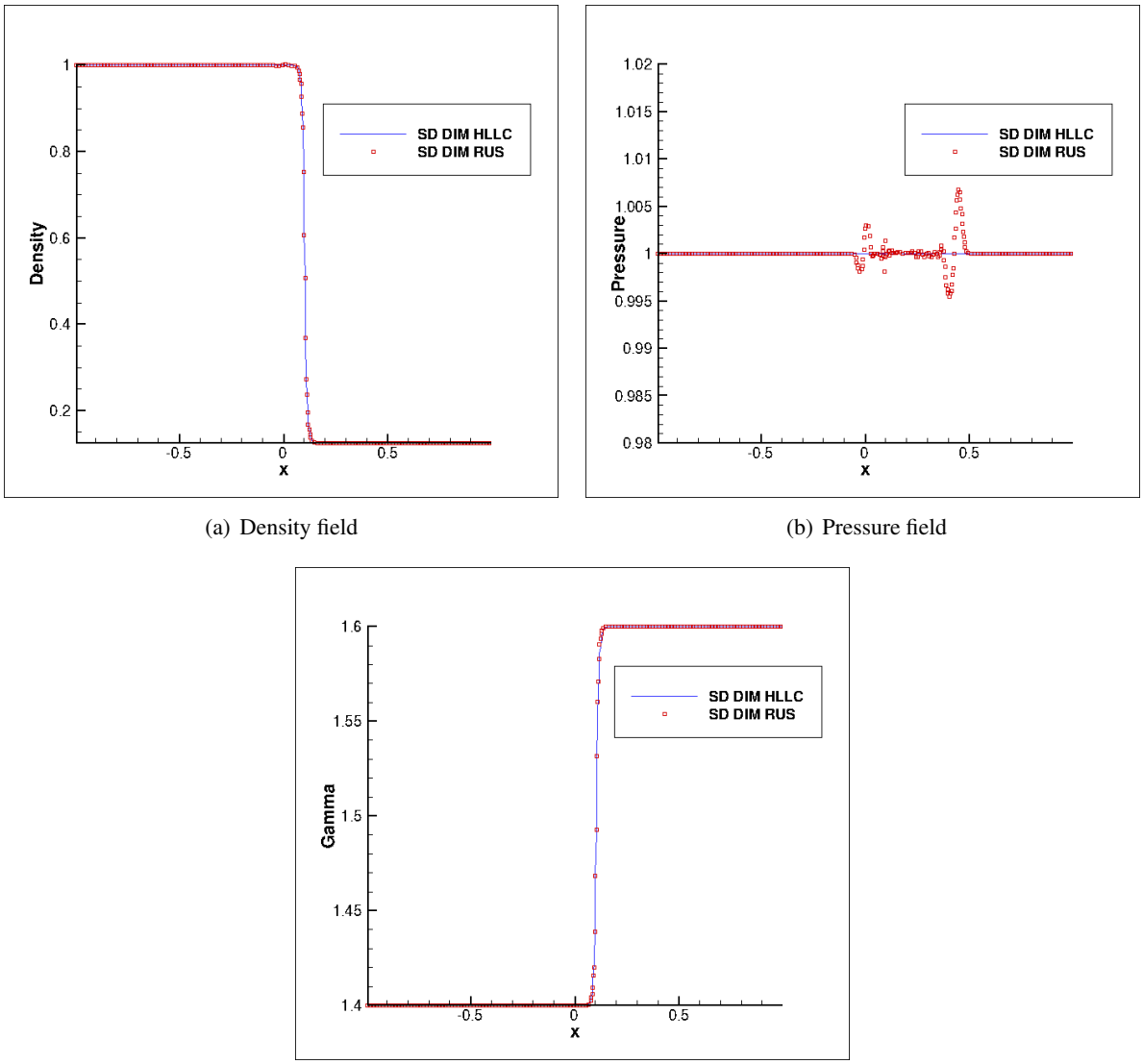
(a) Density field

(b) Pressure field



(c) γ field

Figure 6.5: Sod's shock tube with γ variation ($t = 0.2$ seconds).



(a) Density field

(b) Pressure field

(c) γ field

Figure 6.6: Material front advection (t = 0.1 seconds).

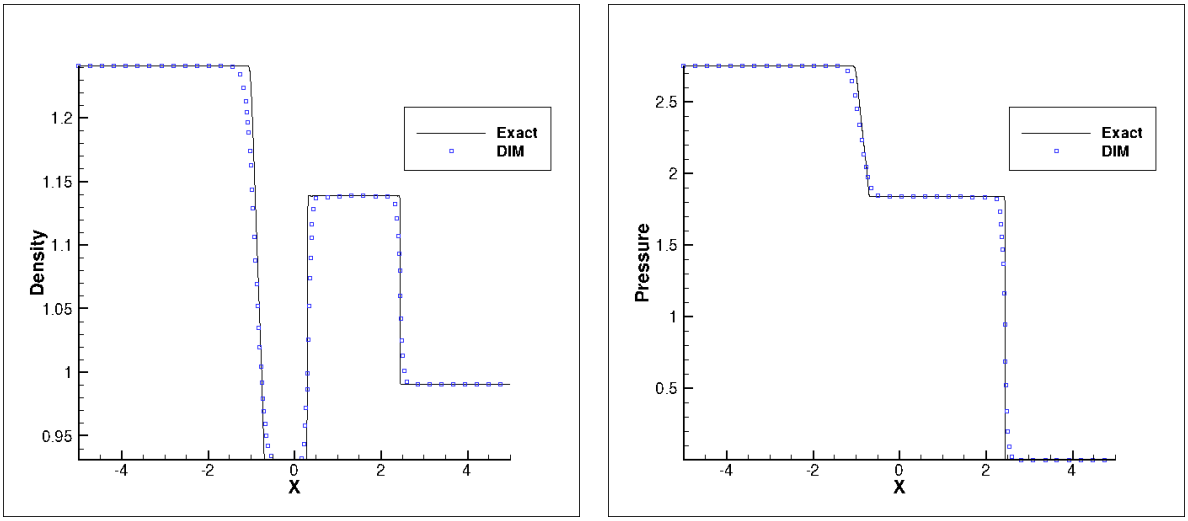
for issues. Figure 6.6 shows the results from density, pressure, and γ. While the density and γ fields (Figure 6.6 (a) and (b)) appear to be captured well, Figure 6.6 (b) shows issues with the Rusanov flux, where pressure oscillations are produced around the traveling interface. In fact, Figure 6.6 (a) shows a small density fluctuation before the material front in the Rusanov flux results. These plots illustrate the necessity of using the HLLC flux for problems which contain multiple materials, as non-physical pressure oscillations can occur at material interfaces.

6.6 Two-Phase Gas-Liquid Problem

The following case demonstrates the method's ability to handle high pressure ratios and different material properties [133]. The domain, $x \in [-5, 5]$, is discretized with 2,500 elements and a P^2 reconstruction is used. The HLLC Riemann solver is applied for the problem due results from the previous test case. The initial conditions are defined as the following:

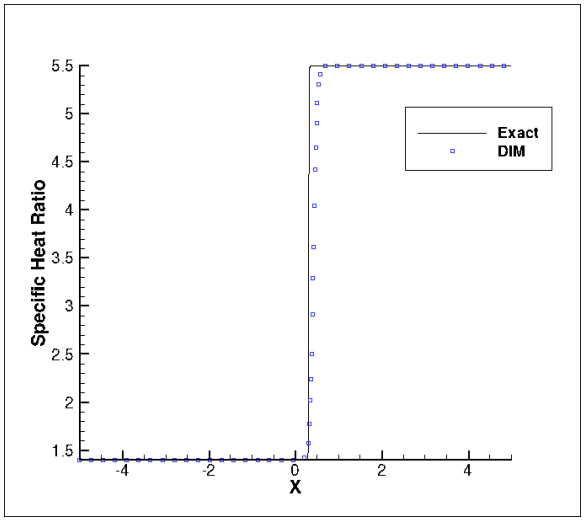
$$(\rho, u, p, \alpha_1)_L = (1.241, 0, 2.753, 1.0), \quad (\rho, u, p, \alpha_1)_r = (0.991, 0, 3.059 \times 10^{-4}, 0.0)$$

These conditions yield an interface pressure ratio of roughly 9000, a high ratio which tests the robustness of the method. The material interface is initially located at $x = 0$ where the left and right materials have differing γ and p_∞ coefficients. The left material is modeled as air under high pressure, with a specific heat ratio of $\gamma_L = 1.4$ and pressure coefficient $p_{\infty,L} = 0.0$. The right material is modeled as water with a high specific heat ratio of $\gamma_R = 5.5$ and pressure coefficient $p_{\infty,R} = 1.505$. Both materials are modeled using two different equations of state, an ideal gas model for the left state and a stiffened gas for the right state. Due to the high pressure associated with the left state, the interface between the materials will advect to the right, causing material mixing. The simulation is completed until a final time of 0.6 seconds with a time-step of $\Delta t = 1.0 \times 10^{-4}$. Figure 6.7 (a) - (c) shows the solution profiles of density, pressure, and γ compared with the exact solution [104] at a time of $t = 0.6$. The computed solution is shown by the symbols, plotted with a 2% distance. The figures demonstrate how well the solution matches with the exact solution. It is important to note that no pressure oscillations at the interface are apparent.



(a) Density profile

(b) Pressure profile



(c) γ profile

Figure 6.7: Two-phase gas-liquid problem at time t = 0.6 seconds.

6.7 Two-Dimensional Riemann Problem

The following case tests the implementation of the numerical method and the slope limiting scheme in multiple dimensions. A square domain, $[0, 0.3] \times [0, 0.3]$, is discretized with 40,000 elements. In each element, a P^2 reconstruction is completed. Like the one-dimensional cases, a interface is defined at $x + y = 0.5$. To the left and right of this interface, the initial conditions are the following:

$$(\rho, u, v, p)_L = (1.0, 0, 0, 1.0), \quad (\rho, u, v, p)_R = (0.125, 0, 0, 0.4)$$

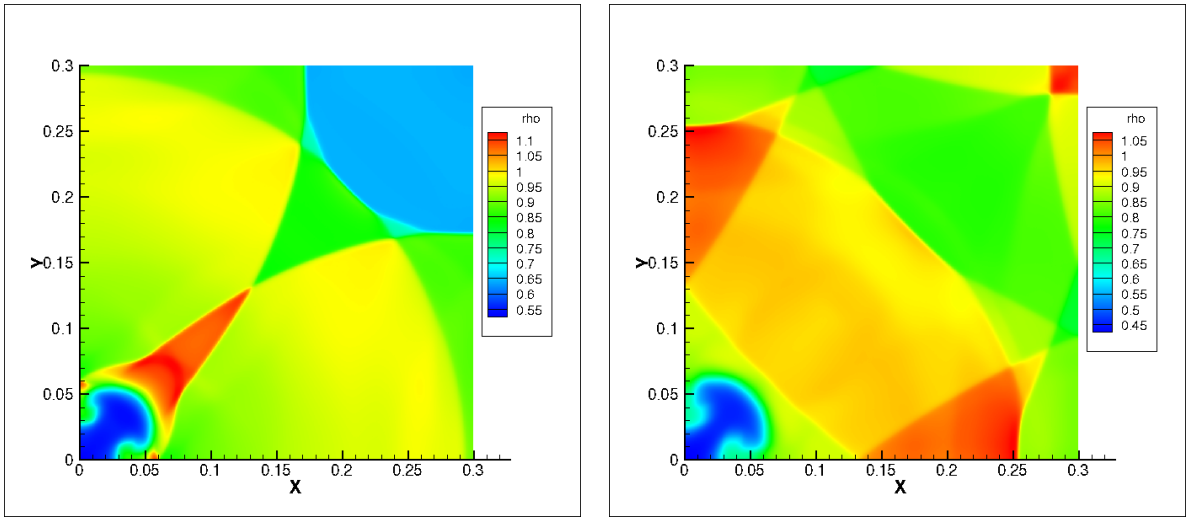
The ideal gas equation of state is used with a constant $\gamma = 1.4$. Along each boundary, a wall condition is imposed, which is defined as

$$\boldsymbol{q}_{wall} = \begin{pmatrix} \rho_L \\ (\rho u)_L - 2\rho_L (\boldsymbol{V} \cdot \boldsymbol{n}) n_x \\ (\rho v)_L - 2\rho_L (\boldsymbol{V} \cdot \boldsymbol{n}) n_y \\ e_L \end{pmatrix}$$

where $\boldsymbol{V} = (u, v)$ is the velocity vector, $\boldsymbol{n} = (n_x, n_y)$ is the face normal vector, and subscripts L indicate the solution at the wall interior. This test case does not have an exact solution, but due to the initial conditions imposed, the solution should demonstrate symmetry about $x = y$. If symmetry is not observed, there exists some numerical problem with the implementation of the method or the slope limiter. The results are shown in Figure 6.8 (a) and (b) at two different times. It should be noted that symmetry about $x = y$ is observed at both times, and throughout the simulation. This indicates that both the method and the slope limiter are indeed implemented correctly.

6.8 Two-Dimensional Blast Wave

This case is identical to the discontinuous problem discussed in Section 4.3, and is outlined here for clarity. A domain of size $[-1, 1] \times [-1, 1]$ is discretized with 160,000 elements. Density and pressure are initialized as (ρ, p) of 1.0 inside a radius of 0.4. Outside the radius, $\rho = 0.125$ and $p = 0.1$. The solution density contours are shown in Figure 6.9 (a) for a P^2 reconstruction. Again, the density is recorded across the centerline, $y = 0$, and plotted with the reference solution from Toro [118]. The purpose of this test case is to investigate the performance increase of utilizing GPUs when compared to



(a) Density field at $t = 0.8$ seconds

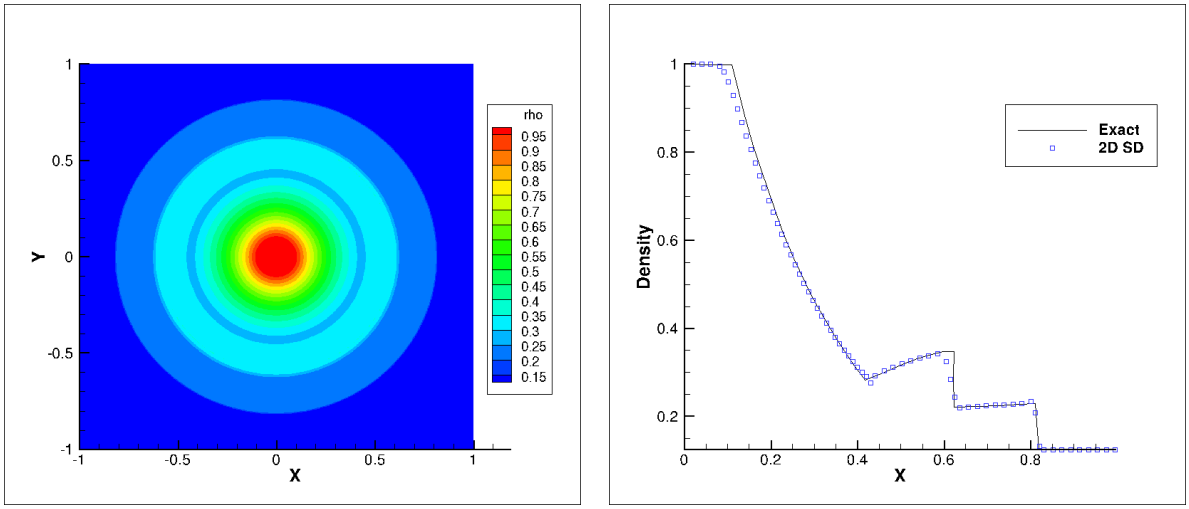
(b) Density field at $t = 1.4$ seconds

Figure 6.8: Two-dimensional Riemann problem results.

Table 6.1: Speed comparisons for two-dimensional SD (time refers to time/iteration).

Hardware	P^1 time	Speed-up	P^2 time	Speed-up	P^3 time	Speed-up
Intel Xeon	2.226	-	3.156	-	5.384	-
NVIDIA K20c	0.019	119.3	0.032	98.9	0.046	117.0
4 × NVIDIA K20c	0.007	341.7	0.010	313.2	0.017	316.7

the CPU code. Table 6.1 shows the time per iteration of three different orders of accuracy for the given mesh. The CPU is a Intel Xeon E5-2640 at 2.50 GHz, while the GPU is a NVIDIA Tesla K20c. The compiler flags discussed in Section 4.1 were implemented with double precision computing for both the CPU and GPU codes. Using only a single GPU, a factor of 100 times increase in computing speed is observed across the orders of accuracy. The GPU workstations employed in this thesis have a total of four GPU K20c cards, and by utilizing all cards an additional 2.5 to 3 times speed-up factor is observed. It is not a linear increase due to hardware limitations, where GPU to GPU communication requires CPU communication first. This implies first transferring memory from the GPU to the CPU, then transferring memory from CPU to CPU, and finally transferring back from CPU to GPU. The memory transfer from GPUs to CPUs is a major bottleneck in GPU computing.



(a) Density contours

(b) Density at centerline

Figure 6.9: Two-dimensional blast wave simulation with P^2 reconstruction ($t = 0.25$ seconds).

6.9 Two-Dimensional Aluminum Impact

In this case, a two-dimensional impact simulation is shown. A pre-shocked and heated semi-infinite aluminum slab is hit by another aluminum slab at ambient pressure traveling at 2 km/s [134]. All variables are non-dimensionalized in the problem. The domain, $[x, y] \in [0, 1] \times [0, 1]$, is discretized with 400 elements. A P^2 reconstruction is completed and the simulation is ran to a final time of $t = 0.04$. The heated aluminum slab is located at $x < 0.5$ with the conditions:

$$(\rho, u, p)_L = (4.0, 0.0, 79.3)$$

while the ambient slab to the right has conditions:

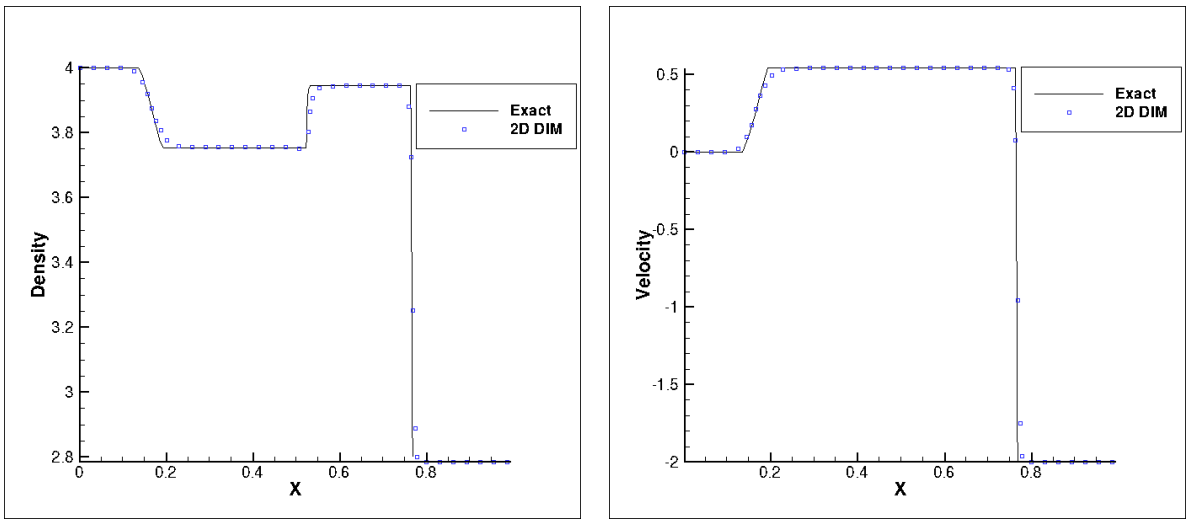
$$(\rho, u, p)_r = (2.785, -2.0, 0.0)$$

Unlike the previous examples, this cases follows the Mie-Grüneisen equation of state (Section 5.3.2), whose parameters for aluminum are given in Table 6.2. At the boundaries, an extrapolation condition is set, written as

$$\boldsymbol{q}_{extrap} = \begin{pmatrix} \rho_L \\ (\rho u)_L \\ (\rho v)_L \\ e_L \end{pmatrix}$$

Table 6.2: Mie-Grüneisen aluminum coefficients.

Parameter	c_0	s_0	ρ_0	e_0	p_0	Γ_0
Value	5.238	1.338	2.785	0.0	0.0	2.0



(a) Density profile

(b) Velocity profile

Figure 6.10: Solution of aluminum impact problem at $t = 0.04$ seconds.

While this case does not require the use of SD-DIM, since both states are modeled by the same equation of state, it does show the application of the solver to handle more complicated equations of state. The density and velocity profiles are shown in Figure 6.10 (a) and (b) for a two-dimensional simulation. The data was gathered along the centerline, $y = 0.5$, and plotted with the exact solution [134]. The results demonstrate the correct implementation of the equation of state and the accuracy of the method.

6.10 Grid Convergence Study

The effects of grid refinement and increasing order of accuracies are studied in this test case. In a domain $[-1,-1] \times [1,1]$, a flat target is defined for $y < 0.0$, while above this the region is defined as air. In a small radius, $r < 0.1$, a small explosive device is initialized within the domain in a specified location. The initial conditions are shown in Table 6.3 for the three materials. The stiffened equation of state is used with $p_\infty = 0$. Three different computational grids were investigated: G_1 (10,000 elements),

Table 6.3: Grid convergence study initial conditions.

Material	ρ	e	γ
Target	20.0	1.0	2.6
Air	1.0	1.0	1.4
Bomb	1.0	100.0	1.4

Table 6.4: Grid convergence study results at $t = 0.2$ seconds.

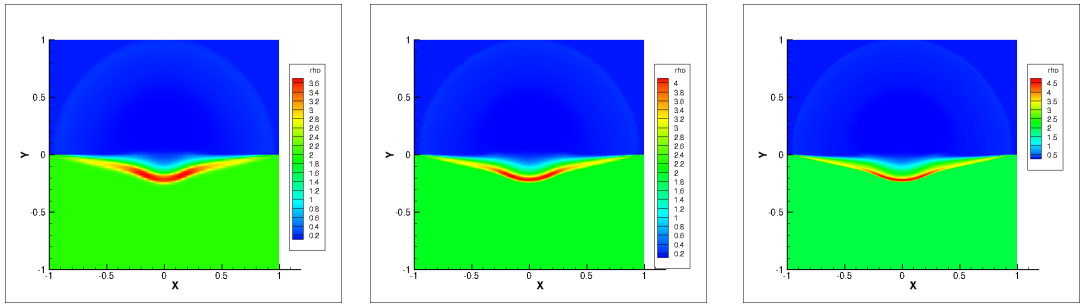
Grid	P^1	P^2	P^3	P^4	P^5
G_1	12.36	11.42	11.16	11.13	11.11
G_2	10.98	11.38	10.27	10.23	10.21
G_3	10.50	10.08	10.03	10.02	10.01

G_2 (40,000 elements), and G_3 (160,000 elements). Two different simulations were completed for each grid and each order of accuracy, one with the explosive device above the surface, and another with the device below the surface. The total kinetic energy transferred to the target surface is monitored for both surface and subsurface simulations, where the total kinetic energy is written as

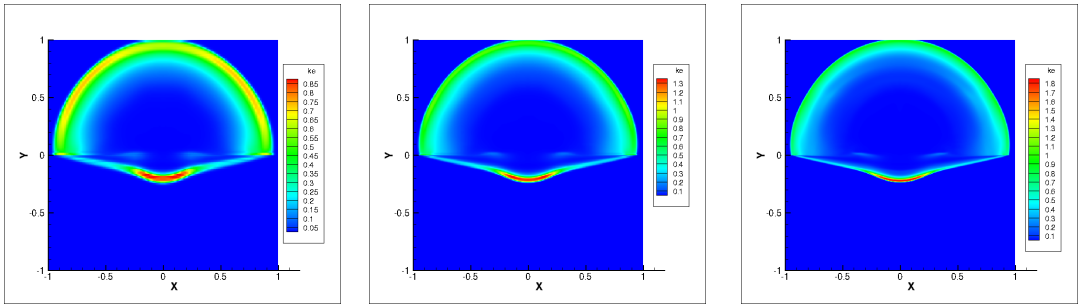
$$e_k = \frac{1}{\Omega} \int_{\Omega} \frac{\boldsymbol{V} \cdot \boldsymbol{V}}{2} d\Omega$$

where Ω is the size of the target body. The energy from the subsurface blast is divided by the energy from the surface blast to yield a coupling factor. This coupling factor is used to investigate the grid convergence.

The energy device center was set at $y = 0.232$ for the surface blast and $y = -0.464$ for the subsurface detonation. The significance of these locations is discussed in a report by the National Research Council (NRC) [135], where at these specific locations, a subsurface detonation is 10 times more efficient at coupling kinetic energy to the target than a surface explosion. The reader is directed to the report for more information. The solutions at two different times are shown in Figures 6.11 to 6.14. Each figure shows a P^2 reconstruction of density and kinetic energy for the three different grids tested. The small structures become more apparent as the grid is refined from G_1 to G_3. The energy coupling factors are



(a) Density G_1 (b) Density G_2 (c) Density G_3



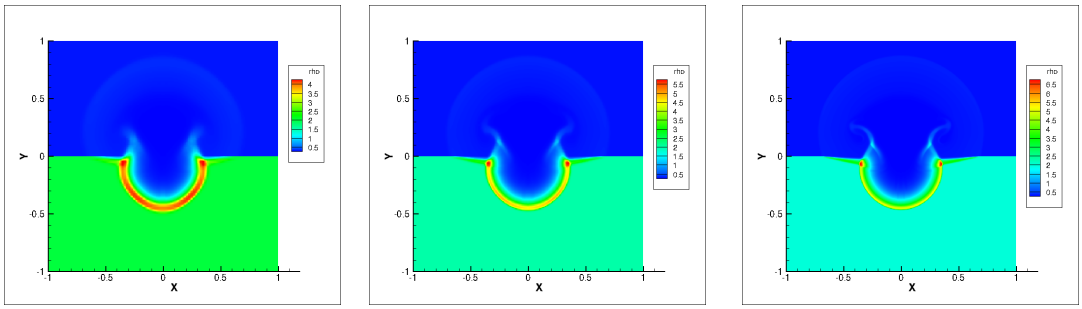
(d) Kinetic energy G_1 (e) Kinetic energy G_2 (f) Kinetic energy G_3

Figure 6.11: Surface explosion at $t = 0.1$ seconds.

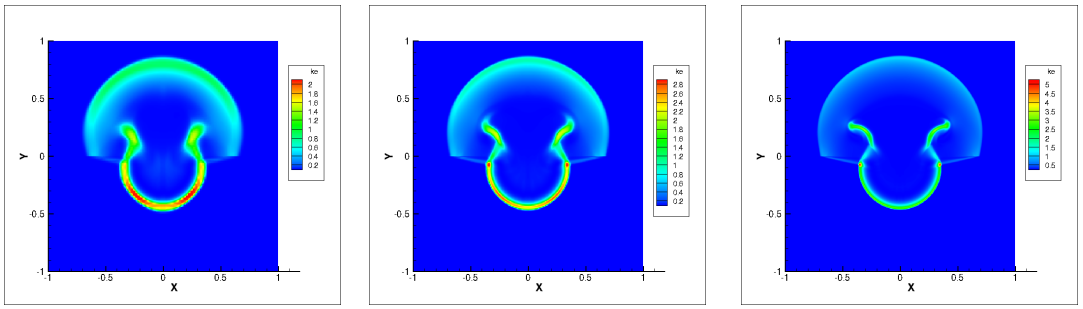
shown in Table 6.4, where through grid and order refinement, the factor of 10 is observed. These results were gathered at $t = 0.2$ s, but results can be obtained at any time after the shock from the surface explosion has contacted the material. Before this time, the energy to the target is zero, and the coupling factor would be infinity. Note that for G_3 with a P^2 reconstruction indicates 9 points per element, which yields 1.4 million degrees of freedom per equation. This result is important as it's the first result which produces an error of less than 1% with respect to the factor of 10 which should be observed. If the grid is too coarse or the order of accuracy is too low, the results greatly over-predict the coupling factor.

6.11 Cross-Code Comparison

In this section, the developed code is compared with the RAGE hydrocode from Los Alamos National Lab [136, 71]. The problem considered is the following: A hypervelocity impactor is to strike an asteroid target traveling at 11.5 km/s. At two different times, the depth of the generated crater is recorded and compared across the codes. The asteroid target is circular with a 50 m radius. The com-

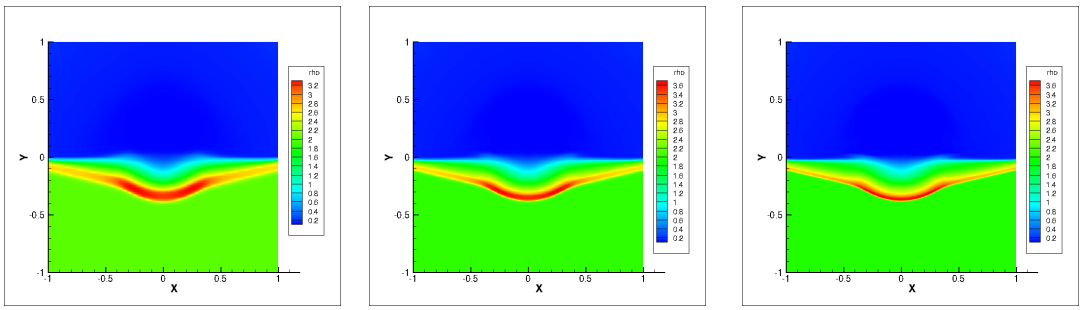


(a) Density G_1 (b) Density G_2 (c) Density G_3

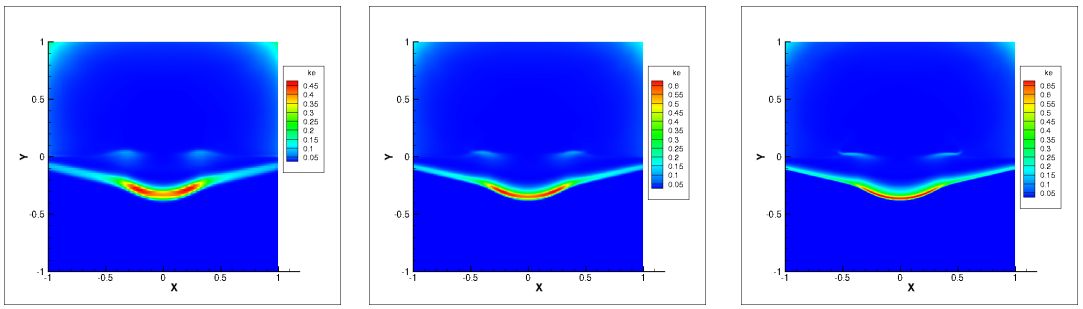


(d) Kinetic energy G_1 (e) Kinetic energy G_2 (f) Kinetic energy G_3

Figure 6.12: Subsurface explosion at $t = 0.1$ seconds.

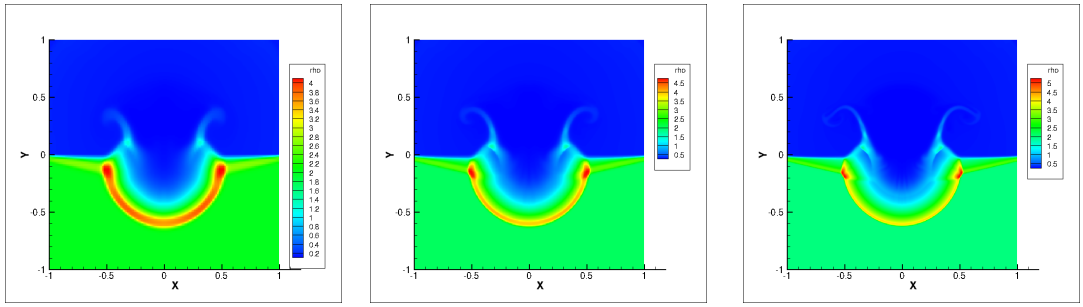


(a) Density G_1 (b) Density G_2 (c) Density G_3

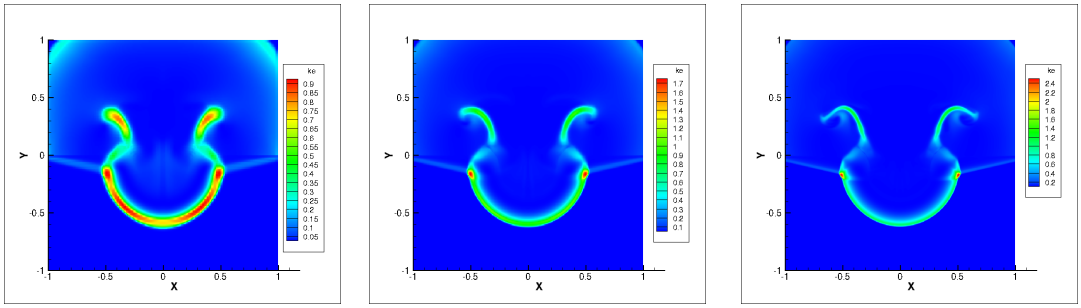


(d) Kinetic energy G_1 (e) Kinetic energy G_2 (f) Kinetic energy G_3

Figure 6.13: Surface explosion at $t = 0.2$ seconds.



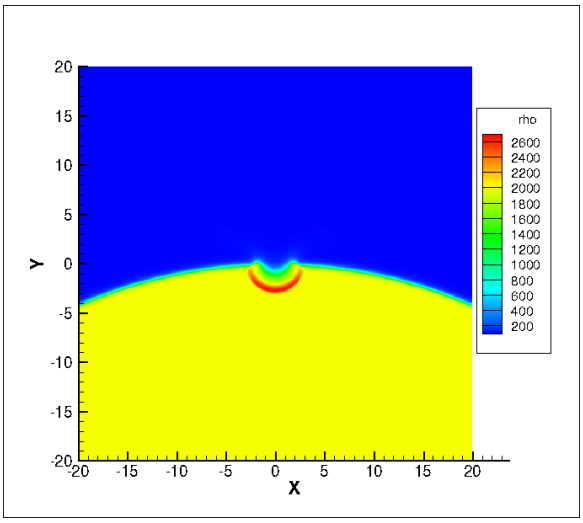
(a) Density G_1 (b) Density G_2 (c) Density G_3



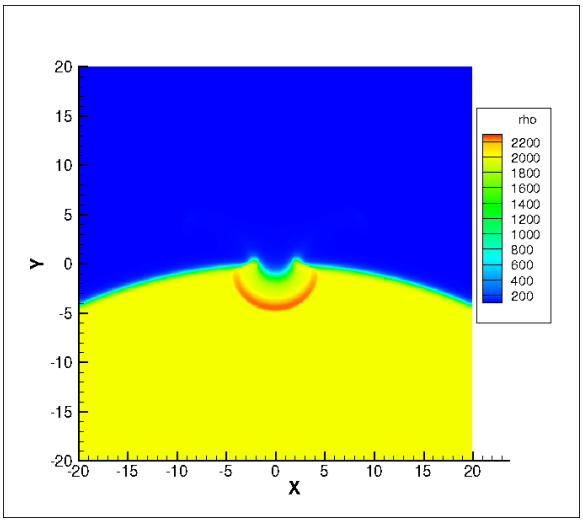
(d) Kinetic energy G_1 (e) Kinetic energy G_2 (f) Kinetic energy G_3

Figure 6.14: Subsurface explosion at $t = 0.2$ seconds.

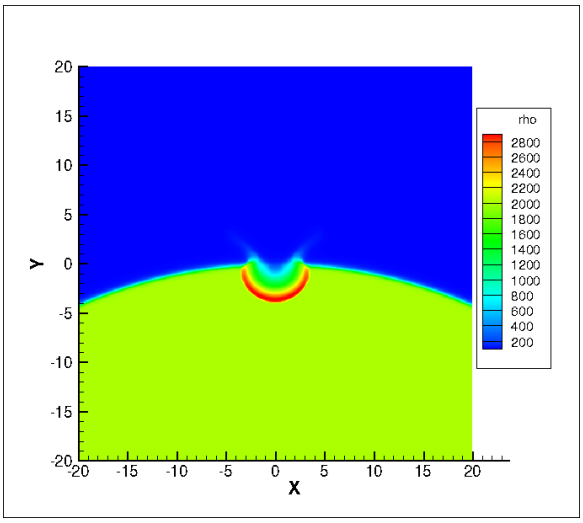
position is assumed to be 100% granite, with a density of 2000 kg/m^3. The impactor is assumed to be an aluminum box, with a varying density of 160 kg/m^3 and 600 kg/m^3 for two seperate simulations. The size of the impactor is 1×1 m, which yields impactors with masses of 160 kg and 600 kg for the two simulations completed. The density contours at two times are shown in Figure 6.15 for the two impactors. Figure 6.15 (a) and (b) show the 160 kg impactor, while Figure 6.15 (c) and (d) show the 600 kg impactor. The computational domain is set as $[-20, -20] \times [20, 20]$ with 400,000 elements and a P^1 reconstruction (1.6 million degrees of freedom per equation) per element to discretize the problem. All materials are modeled using the stiffened equation of state, with the parameters listed in Table 6.5 [109, 137]. Before discussing the results, note that this simulation, and future asteroid simulations, are only two-dimensional. This indicates that the simulations mimic a infinitely long rod impacting an infinitely long cylinder. The impactor leaves a relatively small crater in the asteroid target after impact, which is expected due to the low impacter mass. The crater can be analyzed for post-processing to compare the two computer programs. The depth of the crater is recorded and compared with results from RAGE [138] at a time of 1 ms and shown in Table 6.6. HyperX is the name of the GPU high-order



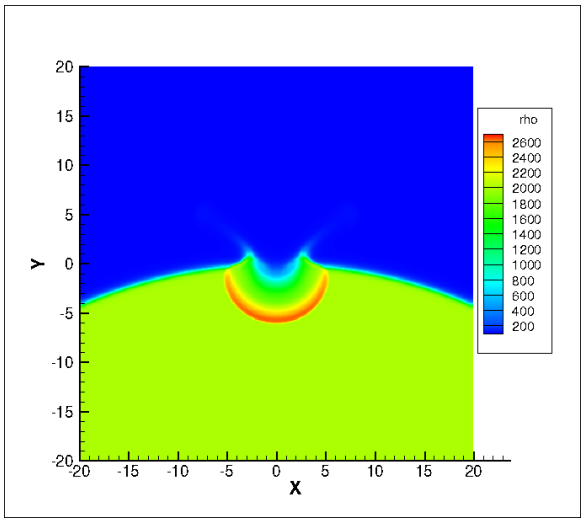
(a) 160 kg impactor at t = 1.0 ms



(b) 160 kg impactor at t = 2.0 ms



(c) 600 kg impactor at t = 1.0 ms



(d) 600 kg impactor at t = 2.0 ms

Figure 6.15: Density contours of crater generation.

Table 6.5: Stiffened gas equation of state parameters.

Parameter	γ	p_∞
Air/Space	1.4	0.0
Aluminum	3.8	1.4×10^9
Granite	2.6	1.42×10^{11}

Table 6.6: Cross-code crater depth comparison results at 1 millisecond.

Solver	162 kg	600 kg
HyperX	0.8 m	2.0 m
RAGE	$>$ 1.0 m	2.2 m

method code developed at Iowa State University (ISU). Results show that while the HyperX solver produces slightly smaller craters than RAGE for the two impactors, good agreement is observed between the results. This gives confidence in the HyperX code for future problems.

CHAPTER 7. ASTEROID DISRUPTION SIMULATIONS

In this chapter, the major results of the work are presented and discussed. A target asteroid is impacted by Kinetic-Energy Impactors (KEIs) and nuclear blast waves. The first simulation set models KEIs against targets with no damage quantification as a preliminary study. The added damage model follows, to contrast the previous results. Finally, the Hypervelocity Asteroid Intercept Vehicle (HAIV), discussed in Section 1.2.2, is simulated to show the effectiveness of nuclear weapons against asteroids.

7.1 Hypervelocity Impactor Vehicles

As previously discussed, nuclear options are the most mass-efficient means for storing energy. However, non-nuclear methods for asteroid disruption should be investigated, for both engineering and political reasons. This section investigates two impactor systems against an asteroid target, and compares the particle dispersal speeds between the two approaches.

7.1.1 Problem Description

Two different KEI systems are investigated in this work. The first system is a single, heavy impactor, deemed the Single Kinetic-Energy Impactor Vehicle (SKIV). The impactor is designed as a square box with a density of 5,000 kg/m^3 and each side 1 m in length. This yields an impactor with a mass of 5,000 kg. One question posed is the following: What if instead of a single hit, a shotgun effect is studied, can this be more effective at disrupting a target? To explore this, the Multiple Kinetic-Energy Impactor (MKIV) system is introduced. The aforementioned SKIV is split into several smaller impactors, with the total kinetic energy between the two systems held constant. In the current study, the SKIV is broken into five seperate impactors to form the MKIV system. For equivalent kinetic energy, each impactor would needs a mass of 1,000 kg and travel at the same speed as the SKIV. The density of each impactor

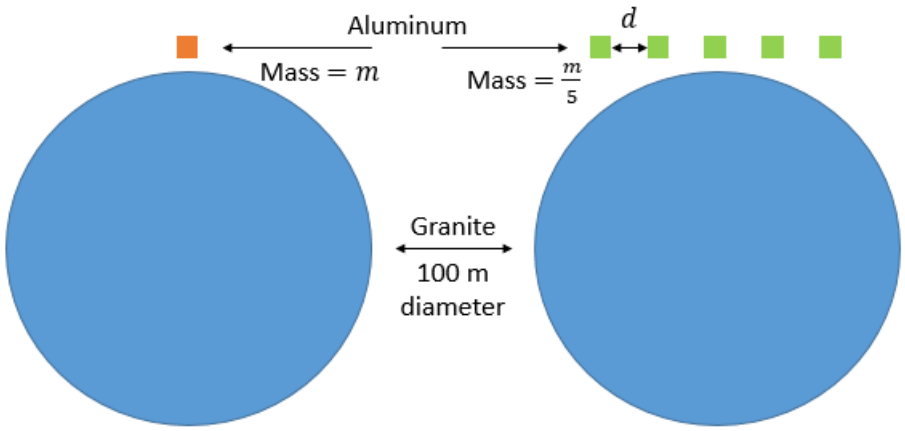


Figure 7.1: The SKIV (left) verses the MKIV (right) system. The orange box is 5,000 kg, while each green box is 1,000 kg. The asteroid target is circular with a diameter of 100 m.

is lowered to 1,000 kg/m^3, and the same square box is used for the design of each impactor. The asteroid target is assumed to be circular with a diameter of 100 m, and has a composition of 100% granite with a density of 2,000 kg/m^3. The overall problem set-up is shown in Figure 7.1 for both impactor systems. The SKIV system (shown in the left of the figure) contains a single orange impactor with mass $m = 5,000$ kg. The MKIV system (shown in the right of the figure) has five impactors, each with a mass of $5,000/5$ kg, or 1,000 kg. The spacing of impactors in the MKIV system is held constant and is defined for each simulation. Each impactor travels at 11.5 km/s, which holds the total kinetic energy constant between each system.

In order to simulate this problem, an equation of state is needed to model each material present. Due to the extremely high pressures associated with the impact from the incoming mass at the specified velocity, the stiffened equation of state (Section 5.3.1) will be used to model the problem. The coefficients for each material are given in Table 6.5. Doing a quick calculation, one can find the speed of sound in granite to be c = 4,296.5 m/s using the coefficients in Table 6.5, a pressure of p =101,325 Pa, and a density of 2,000 kg/m^3. Since each impactor strikes the target at 11.5 km/s, well over twice the speed of sound in granite, the impactor speed is truly hypervelocity. In addition, three distinct materials are required to be modeled: Space, aluminum, and granite. Due to the limitations of the solver, empty space with zero density is not allowed. Hence, the outside space is modeled as air with a low density. In this work, the density is set to 10 kg/m^3 for numerical stability purposes (densities lower can cause

Table 7.1: Averaged velocity dispersion speeds at time $t = 0.06$ seconds.

	SKIV	MKIV
$\|\boldsymbol{V}\|$ (m/s)	23.31	30.52

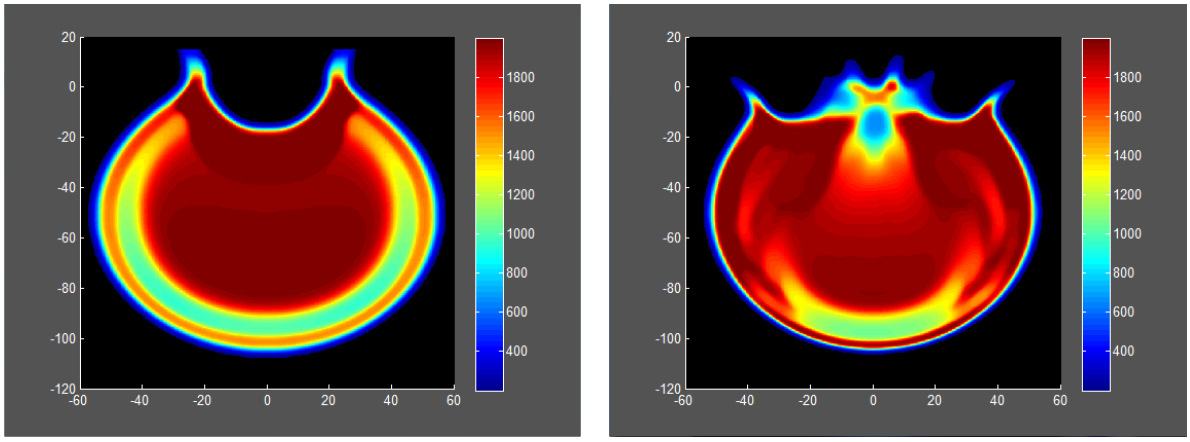
instabilities in the results). The system to be solved is given in Equation (5.2.9), where three equations are needed for conservation of mass, two equations for conservation of momentum, one equation for conservation of energy, and three additional equations for the volume fractions. This yields a system of nine partial differential equations to be solved at every point in the domain.

7.1.2 Preliminary Simulations

The computational domain is $[x, y] \in [-65, -115] \times [65, 15]$ and partitioned with 250,000 elements with a P^1 reconstruction within each element. There is no damage model integrated for these simulations, which makes predicting fragment sizes or showing disruption of the asteroid target impossible. Hence, these simulations should only be viewed as preliminary results. Simulations are completed until a computational time of $t = 0.06$ seconds with a small time-step of $\Delta t = 3.0 \times 10^{-6}$. For the MKIV system, each impactor was set 10 m apart.

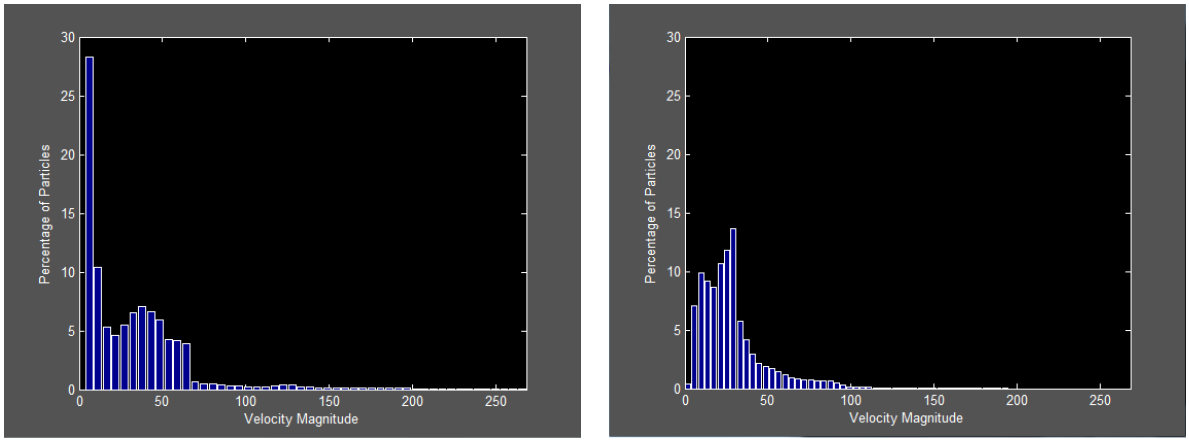
Figure 7.2 shows the simulation results without a damage model for the SKIV and MKIV systems. Figures 7.2 (a) and (b) illustrate the density contours of the two configurations, where regions of high density are observed which may correlate to large fragments. One important result is the relative size of these regions, where a large high-dense region is observed in the SKIV results near the core of the target. Furthermore, when compared against the MKIV density results, differences are observed in the size and locations of the high-dense regions. The major difference in the core is of high interest, as this may indicate damaged material, no longer intact.

The velocity histograms are plotted in Figures 7.2 (c) and (d) for the systems. These results indicate that the SKIV system has a large percentage of particles traveling slower than the MKIV system. This can be attributed to the core of the target in the SKIV case, where the large high-dense region shows no displacement. The velocity histograms are averaged and shown in Table 7.1. The MKIV system is more



(a) Density contours (kg/m^3) SKIV

(b) Density contours (kg/m^3) MKIV



(c) Velocity histogram (m/s) SKIV

(d) Velocity histogram (m/s) MKIV

Figure 7.2: Simulation results of SKIV and MKIV at time $t = 0.06$ seconds (no damage model).

effective at disrupting the asteroid target, since the velocity dispersion speeds at roughly 31% greater than the SKIV dispersal speeds.

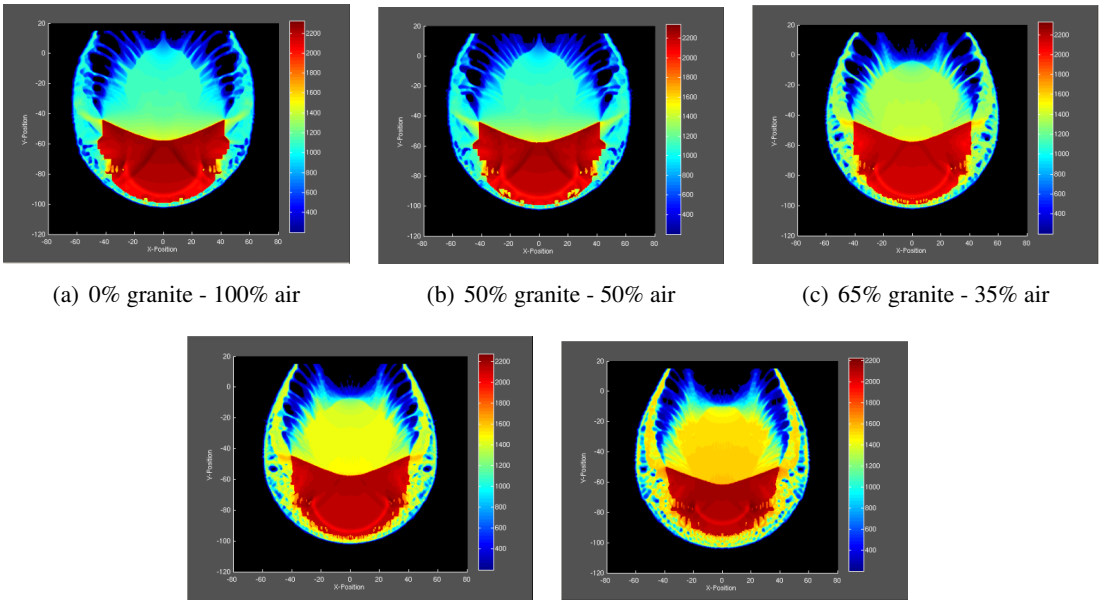
While these results do not yield fragments or show pulverization of the target, they serve as a benchmark for moving forward when a damage model is integrated to the simulations. One important result is the crater size for each case, which can be used to help tune the damage model in the next section.

7.1.3 Damage Model Integration

As discussed in Section 5.4, a damage model is required in order to simulate asteroid break-up and determine if fragmentation will occur. Since the asteroid target is assumed a granite composition, the results from Stowe [129] will be used, where the maximum compressive strain is $\epsilon_c \approx 2000.0\mu$ and the maximum tensile stain is $\epsilon_t \approx 200.0\mu$. Once damage has been detected, a phase change is completed at the damage location. To investigate the effects of different phase changes, the SKIV simulation is completed as introduced in Figure 7.1. The simulation is stopped at a time of $t = 0.06$ s with a time step of $\Delta t = 1 \times 10^{-5}$. Again, the domain is $[x, y] \in [-65, -115] \times [65, 15]$ and partitioned with 250,000 elements with a P^1 reconstruction. A parameter study is completed in Figure 7.3, where five different phase ratios are shown. Figure 7.3 (a) represents a mixture where the damaged target looses all of its material properties and behaves like air. In this simulation, the crater, shown in Figure 7.2 (a), is completely lost. Damaged material travels upward, in the positive y-direction at the impact location. It is more physically reasonable to still observe the crater in this simulation, but even changing the mixture to 50% granite and 50% air, shown in Figure 7.3 (b), shows very little change in crater formation or damage amount. Only when the granite composition is above 60% does crater formation take place. Further increasing the phase ratio in Figures 7.3 (d) and (e) yield a better crater profile with comparable damage between the results. For the remainder of this thesis, a 75% granite and 25% air composition is assumed for the damaged material, as shown in Figure 7.3 (e). Physically, this indicates that the damaged material has lost 25% of its original stiffness and behaves more like a rubble pile. It should be noted that the compressed region within the target, apparent in all phase mixtures, will most likely undergo additional damage due to shear waves, which are not presently modeled.

7.1.4 Kinetic Impactor Results

The computational domain is $[x, y] \in [-65, -115] \times [65, 15]$ and simulations are completed across two grids and three orders of accuracy. The coarse mesh, G_1, is partitioned with 250,000 elements while the refined mesh, G_2, has 562,500 elements. Table 7.2 shows the total number of Degrees of Freedom (nDOFs) and time to complete a simulation for the impactor problems. The nDOFs is defined as the total number of points multiplied by the total number of partial differential equations solved at each



(a) 0% granite - 100% air (b) 50% granite - 50% air (c) 65% granite - 35% air

(d) 70% granite - 30% air (e) 75% granite - 25% air

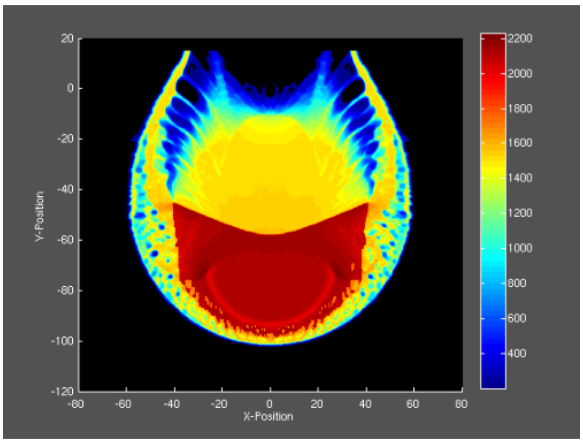
Figure 7.3: Density contours of different phase mixtures for granite and air (t = 0.06 seconds).

Table 7.2: Total simulation nDOFs and time on four K20 GPUs.

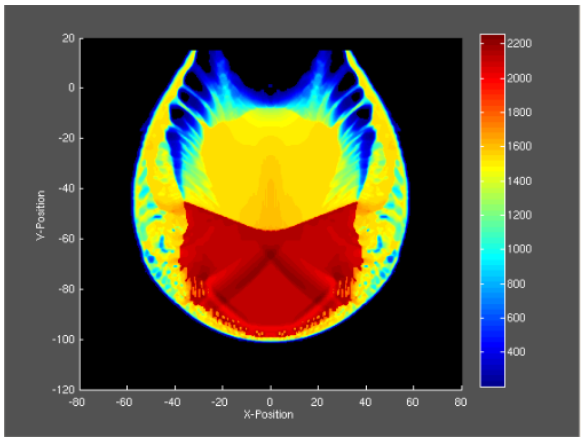
	G_1 nDOFs (million)	G_1 time (min)	G_2 nDOFs (million)	G_2 time (min)
P^1	9.0	5.00	20.25	28.50
P^2	20.25	6.97	45.56	38.07
P^3	36.0	10.43	81.0	59.90

point. In the example of the G_1 grid with a P^3 reconstruction, there are 250,000 elements with 16 points within each element and nine partial differential equations solved at each point. The total time to finish a simulation on four NVIDIA K20 GPUs is given in minutes. Even on a fine grid and high order of accuracy, the simulation can finish in just under an hour.

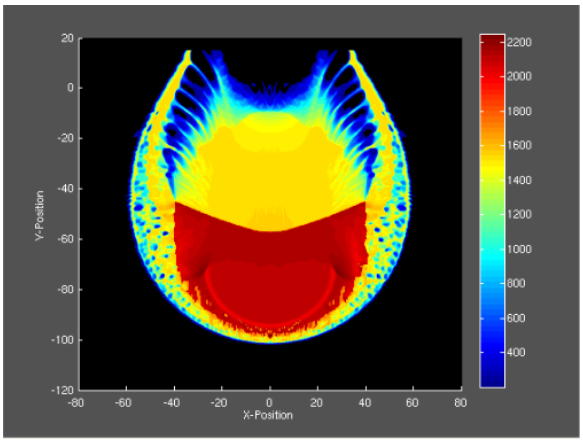
The SKIV system is shown using the two grids and two reconstruction orders in Figures 7.4 and 7.5. As the grid and order is refined, the small structures become more apparent and are less smeared in the density contours. Even by increasing the order of accuracy on the coarse mesh in Figure 7.4 (b), more features are captured in the core of the target. The highest refinement, Figure 7.4 (d), improves upon those features.



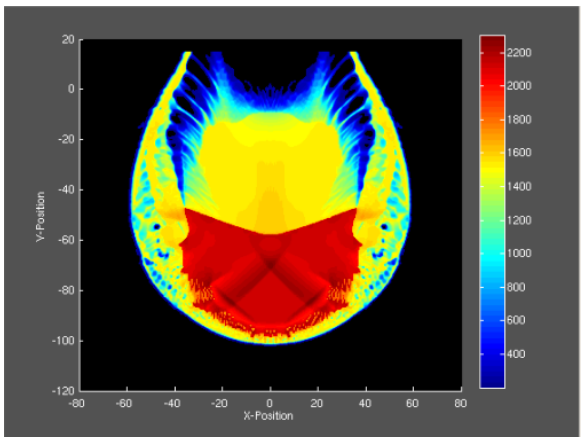
(a) G_1 and P^1 reconstruction



(b) G_1 and P^3 reconstruction

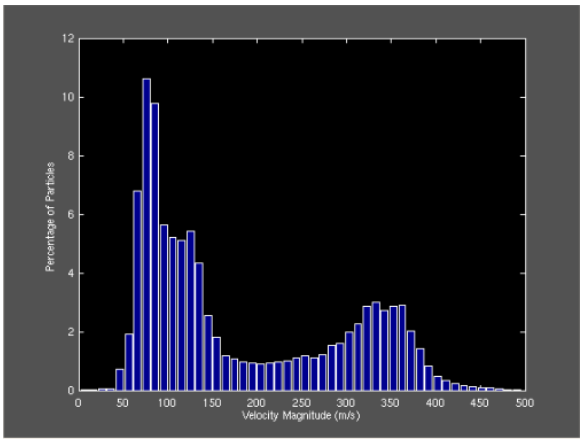


(c) G_3 and P^1 reconstruction

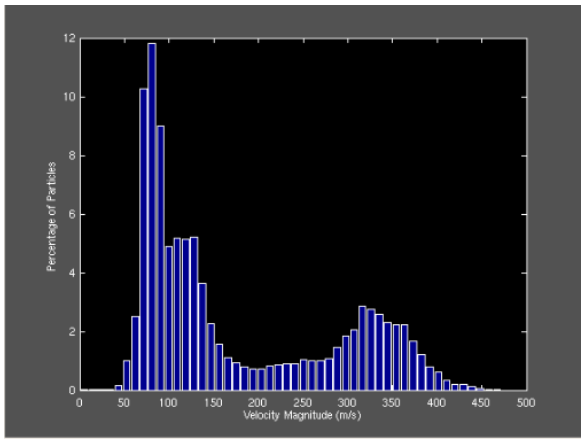


(d) G_3 and P^3 reconstruction

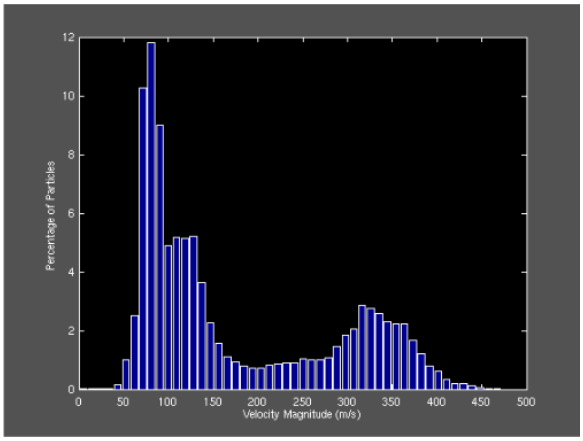
Figure 7.4: Density contours of grid and order refinement for SKIV system (t = 0.06 seconds).



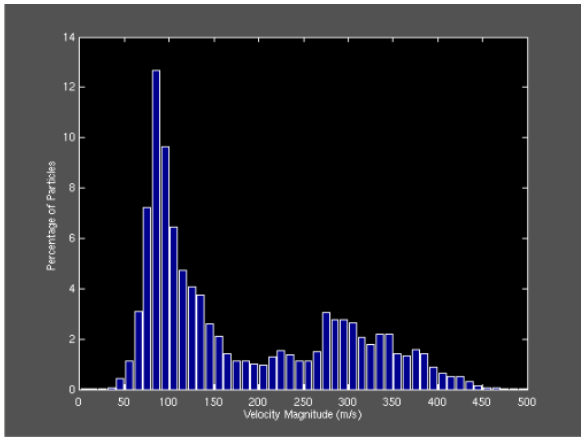
(a) G_1 and P^1 reconstruction



(b) G_1 and P^3 reconstruction

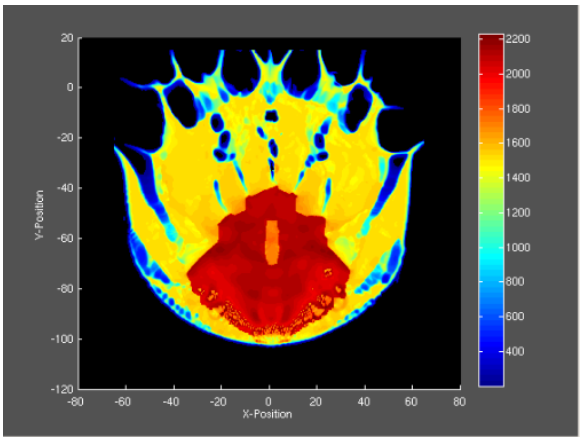


(c) G_3 and P^1 reconstruction

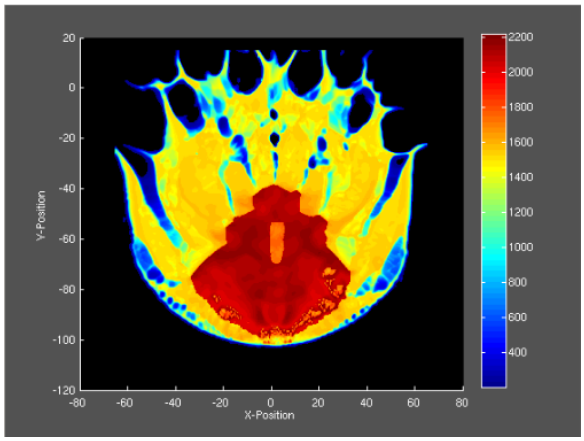


(d) G_3 and P^3 reconstruction

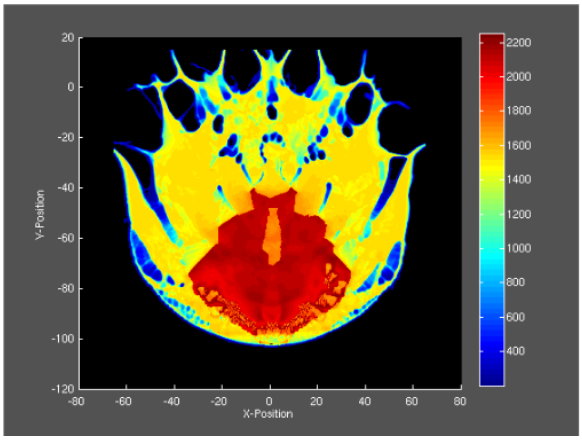
Figure 7.5: Velocity histograms of grid and order refinement for SKIV system ($t = 0.06$ seconds).



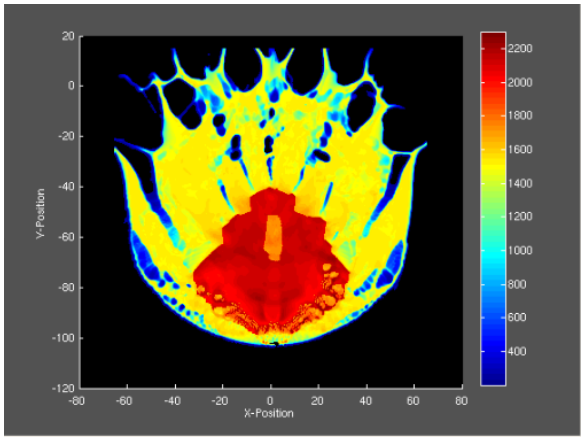
(a) G_1 and P^1 reconstruction



(b) G_1 and P^3 reconstruction

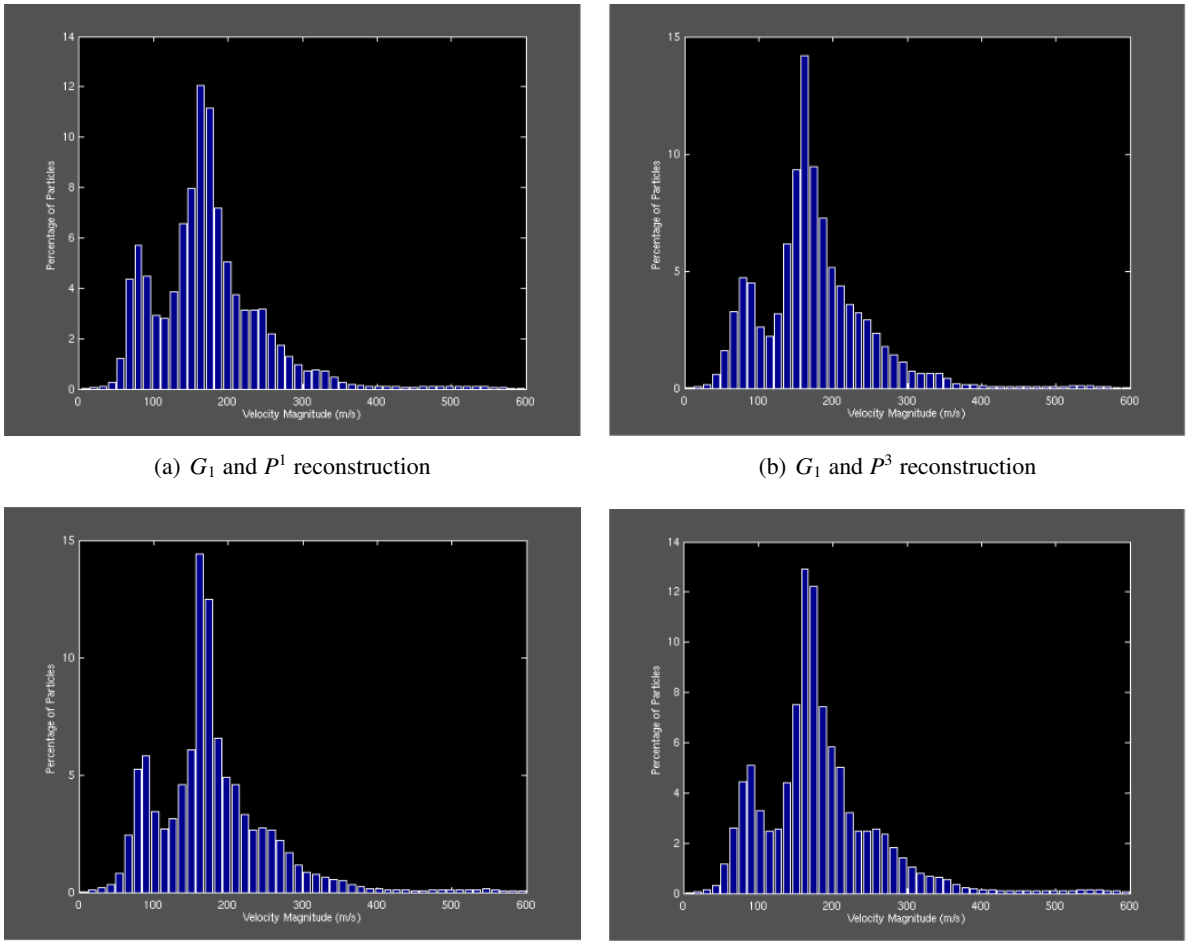


(c) G_3 and P^1 reconstruction



(d) G_3 and P^3 reconstruction

Figure 7.6: Density contours of grid and order refinement for MKIV system ($t = 0.06$ seconds).



(a) G_1 and P^1 reconstruction

(b) G_1 and P^3 reconstruction

(c) G_3 and P^1 reconstruction

(d) G_3 and P^3 reconstruction

Figure 7.7: Velocity histograms of grid and order refinement for MKIV system ($t = 0.06$ seconds).

Table 7.3: Average SKIV system dispersal speeds from grid and order refinement (t = 0.06 seconds).

	G_1	G_2
P^1	179.64	174.67
P^2	168.49	179.95
P^3	171.05	181.40

The velocity histogram plots are observed in Figure 7.5. Regardless of order of accuracy or grid level, the histogram results compare well. A high percentage of particles are traveling near the 100 m/s region, with an additional peak (though much smaller) near the 300 m/s region. Perhaps a more interesting result can be found by averaging the histogram results, as shown in Table 7.3. The coarse mesh results appear to converge near 170 m/s, while the refined mesh results converge roughly 10 m/s higher, near 180 m/s.

The same simulations were completed for the MKIV system on two computational grids at two orders of accuracy. The density contours are observed in Figure 7.6. The major features are consistent throughout the plots, where significant target damage is shown at the impactor locations, major damage is observed around the $y = -40$ m line, and a high-dense region is observed indicated by the red contours. The high-dense region is significantly smaller in the MKIV results when compared to the SKIV results. In addition, more damage is observed in the MKIV density contours, where the asteroid appears to internally fracture. As a second comparison, consider the velocity histogram plots, shown in Figure 7.7. All the plots illustrate a peak particle percentage around 175 - 180 m/s, which is significantly higher than the results in Figure 7.5, where the peak occurs just below 100 m/s.

To compare between the SKIV and MKIV results, the two computational grids are tested at various orders of accuracies. Due to the distributed surface damage in Figure 7.6 a large portion of asteroid surface is ejected in the positive y-direction, leaving the domain through the boundary condition at a time of 0.06 seconds. The time was stopped at 0.03 seconds, where all the asteroid material is still confined within the domain. The results are shown in Table 7.4, where the MKIV has faster dispersal speeds than the SKIV system for all simulations. On the finest mesh and highest order of accuracy, a 7.37% increase is observed in the MKIV system results. These results give further confidence that the

Table 7.4: Average dispersal speeds (m/s) for the SKIV and MKIV systems ($t = 0.03$ seconds).

P^1	SKIV	MKIV	Percentage Increase
G_1	111.91	122.78	9.71
G_2	105.25	116.29	10.49
P^2			
G_1	99.18	116.62	17.6
G_2	108.44	117.15	8.03
P^3			
G_1	99.82	115.34	15.5
G_2	111.30	119.50	7.37

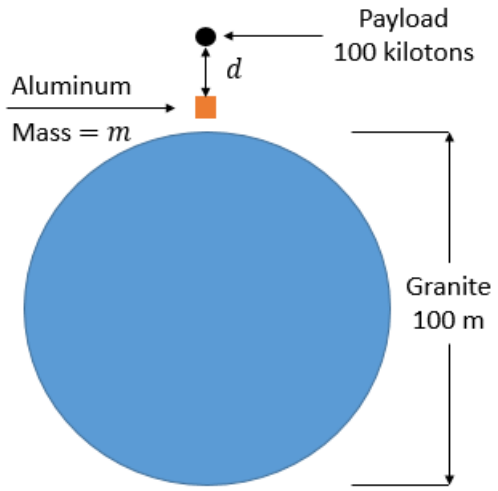
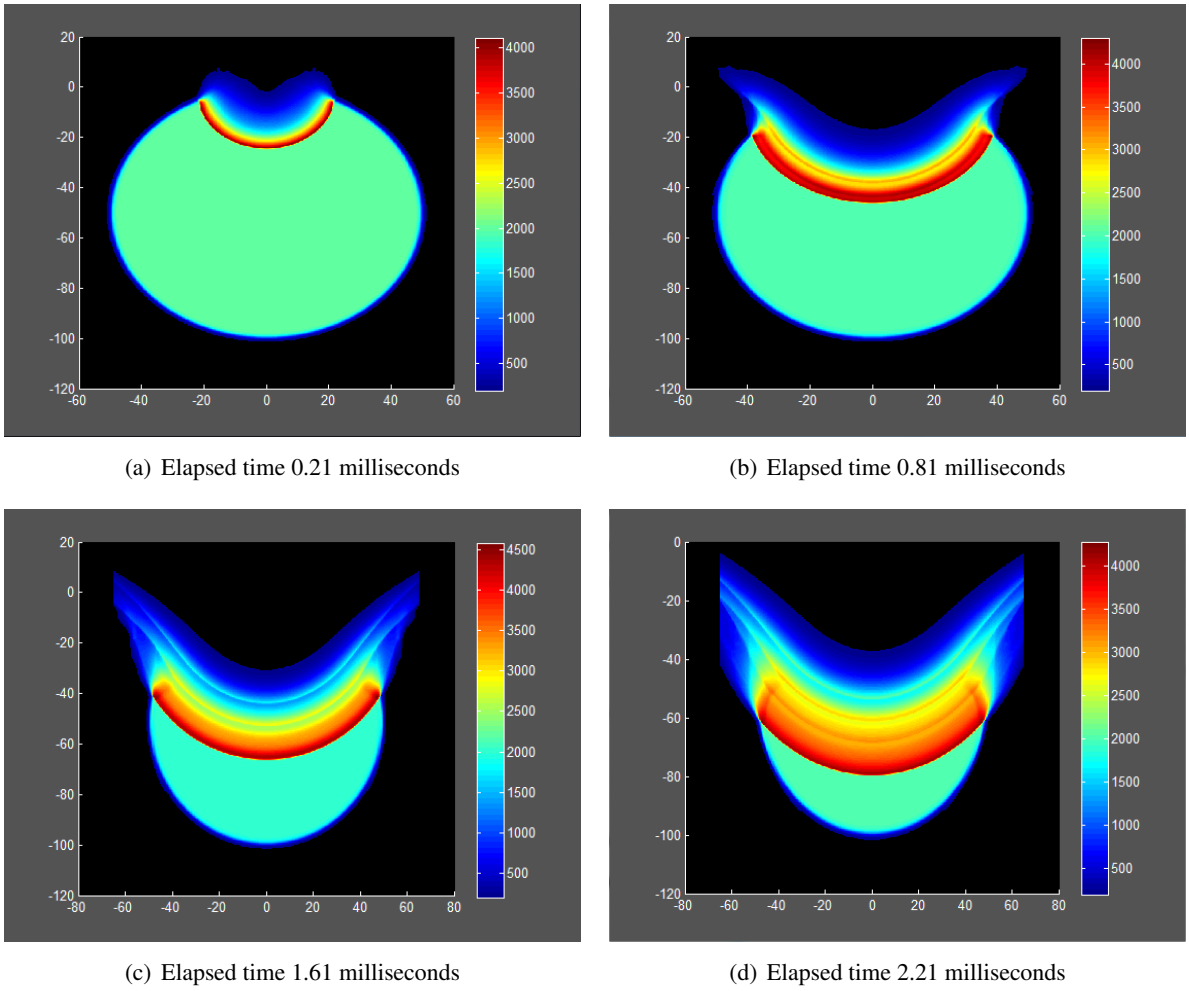


Figure 7.8: HAIV concept illustration.

MKIV system is more effective at pulverizing the asteroid.

7.2 HAIV Results

In this section, a single simulation is carried out to illustrate the effectiveness of the HAIV approach. A computational domain is discretized with 250,000 elements and P^2 reconstruction is applied. The computational domain is $[x, y] \in [-65, -115] \times [65, 15]$, the same domain used in the previous simulations. Figure 7.8 outlines the problem geometry. The impactor is a 1.0×1.0 m aluminum box with a density of 500 kg/m^3 and a mass of 500 kg. The nuclear payload contains 100 kilotons of energy



(a) Elapsed time 0.21 milliseconds

(b) Elapsed time 0.81 milliseconds

(c) Elapsed time 1.61 milliseconds

(d) Elapsed time 2.21 milliseconds

Figure 7.9: Density contours (kg/m^3) for HAIV simulation at specified total times.

(1 kiloton is equal to 4.184×10^{12} joules) and follows the impactor at a distance of 10 m. In order to properly simulate this case, the 500 kg impactor is first simulated until a time of 0.1 milliseconds. At this time, a small crater is formed in the asteroid target. The nuclear device is initialized inside this crater, and the simulation is restarted. The device is treated as an ideal gas with a constant specific heat ratio of $\gamma = 1.4$ and a density of 1.0 kg/m^3. There is no modeling for radiation loss, which indicates this is mostly an ideal case.

Figure 7.9 shows the density contours of the HAIV simulation at specified times. The energy from the nuclear device is distributed into a high pressure mechanical shock wave that contacts the asteroid target within the crater. It should be noted that a portion of the shock is lost to the outside space through

the crater opening. Figure 7.9 (a) shows results at a total time of 0.21 milliseconds, which is 0.11 milliseconds after the nuclear device has detonated. All results show complete target vaporization. A quick calculation can show that resultant compressive strain in the target is significantly higher than the targets maximum allowable compressive strain. With a shock front of around 4,000 kg/m^3, the resulting strain is computed as

$$\epsilon = \sqrt{\frac{\rho_0}{\rho} - 1} = \sqrt{\frac{2000.0}{4000.0} - 1} \approx 30 \times 10^4 \mu \text{ strain} \tag{7.2.1}$$

which is 150 times larger than the maximum compressive strain in granite used in this thesis. This large amount of material deformation clearly indicates vaporization of the target body.

CHAPTER 8. CONCLUSIONS AND FUTURE WORK

This thesis presents the development of GPU-based computational multifluid models to explore aspects of asteroid pulverization and fragmentation. Several high-order numerical methods were optimized and coupled with material interface modeling for implementation on GPUs in order to better explore complex multifluid applications. These methods were compared with GPU CUDA programming to determine the most computationally efficient and accurate method. This thesis establishes that standard and extensively used Finite Volume (FV) method is the least computationally efficient and accurate for a significant number of smooth problem cases. The FV method yields higher errors for a given work unit, while high-order methods produce much lower errors for the same work unit. Additionally, the work proves simulations run with high-order methods are faster than the standard FV method for a given number of degrees of freedom. It was also demonstrated that the Spectral Difference (SD) method is able to produce both lower errors and solutions faster than arguably the most efficient high-order method to-date, the Correction Procedure via Reconstruction (CPR) method.

For discontinuous problems, all methods demonstrated good agreement with a reference solution. Presented results indicate that the CPR method is the computationally cheapest method per iteration for the grids and orders of accuracy tested. The FV method, however, produced the lowest total work unit to complete a simulation for all tests. For higher reconstruction orders, the SD method was again the best performing high-order method, producing solutions only 25% slower than the FV method and just slightly faster than the CPR method. From the results presented in this thesis, it is clear that the SD method is the most efficient method per work unit and one of the computationally cheapest high-order methods with GPU CUDA computing. Therefore, the SD method was selected for the multifluid modeling problem addressed in this thesis.

Presented here is the first application in available literature of coupling the high-order SD method with the Diffused-Interface Method (DIM), which was done to properly simulate multi-material inter-

faces in multifluid problems. The combined framework of SD and DIM was outlined, numerical issues were addressed, and it was verified through numerical simulations that multi-material problems can be resolved accurately. The coupling of SD with DIM and the GPU computing implementation creates a novel computational tool which is well suited to many multi-component problems.

The novel SD-DIM computational tool is applied here to explore the complex, difficult problem of asteroid fragmentation and pulverization from both kinetic impactors and nuclear explosive devices. For non-nuclear methods, this work compared a single kinetic impactor (SKIV) to a multi-bodied system (MKIV) to determine which approach is more effective. Results establish that for an equivalent amount of kinetic energy, the MKIV system is more effective at pulverizing and fragmenting the target asteroid. For all cases considered, the MKIV system produced velocity dispersal speeds 7 - 17% faster than the SKIV system. Observed target damage was significantly more severe for the MKIV case as estimated from density contours, which also indicate improved target disruption. For nuclear methods, a kinetic impactor was blended with a nuclear explosive (HAIV) to maximize the coupling energy from the nuclear device to the target, which resulted in complete target destruction. The nuclear device produced material strains 150 times larger than the materials maximum allowable compressive strain, resulting in immediate and catastrophic failure.

The presented results are for one and two-dimensional problems, and extension to three dimensions is needed. Computational bottlenecks shift when progressing from two-dimensional to three-dimensional problems, as parallelism and coupling per element both increase drastically. It is uncertain if the presented two-dimensional GPU efficiency and accuracy results will hold for three-dimensional problems. However, the two-dimensional approach developed provides a foundation for a three-dimensional study.

The equations of state and asteroid target composition definitions should be addressed for further investigation. Results presented implemented the stiffed gas equation of state for the impactor and target materials. This state equation does not completely describe the materials responses, and more accurate equations of state should be explored. Additionally, asteroid targets are not entirely granite composition, which was assumed here. Asteroids can be composed of ice, iron, granite, and a multitude of other materials. Further exploration is needed for asteroid targets of varying material composition and shapes.

The damage model presented demands further verification with simulation and experimental results, as phase mixtures were tuned to yield physically reasonable solutions. However, comparing with existing results in literature proves difficult, as most materials have yet to be fit to the equations of state presented, further implying the need for more equations of state to be implemented. Additionally, many literature results are presented in three-dimensions, another limitation of the two-dimensional model developed.

It is important to note that the multifluid model developed here is only applicable when the P-wave particle displacement is sufficiently large when compared to the maximum compressive strain in the target material to cause compressive failure and prevent S-wave propagation. The model also breaks down when considering the fragmentation of the body. Both concerns can and should be addressed with a solid dynamics model. The S-waves can contribute to material damage and overall material response, important for modeling. In addition, the individual fragments and inter-fragment collisions cannot be captured with a fluid dynamics model. Finally, further exploration of the transition region between fluid and solid models is needed.

While SD-DIM is capable of solving a diverse set of problems, several important questions arising from the complex nature of the specific application presented in this thesis still exist. Questions as to the validity of the fluid model assumption need to be addressed. Experimental data that would allow validation of the results presented herein is lacking in available literature, and it is recommended that validation and verification be completed before future work continues. Further investigation into the implementation of damage models is still possible and should be pursued. Suitable equations of state that will allow for accurate responses from generic materials are not readily available, but should be derived and integrated into SD-DIM. Hypervelocity phenomena are difficult to capture analytically and the resulting shock waves are difficult to resolve numerically. To address this issue, this work utilizes a simplified fluid model, but a more accurate solid dynamics model is required. Future work in this area should consider and address these issues.

APPENDIX A. DERIVATION OF COEFFICIENTS

This appendix describes the derivation of different coefficients needed in Chapter 2 for each numerical method.

CPR Coefficients

For the Correction Procedure via Reconstruction (CPR) method, two sets of coefficients are needed, one set to take derivatives, and another to provide corrections at element interfaces. Let l define a Lagrange polynomial basis where a degree k polynomial is written as

$$l_j(x) = \prod_{s=1, s\neq j}^{k+1} \left(\frac{x - x_s^l}{x_j^l - x_s^l} \right)$$

where x^l are Gauss-Lobatto points. An example P^2 Lagrange polynomial calculation is shown in Table A.1. The derivative coefficients are found by taking the derivative at each point with respect to each Gauss-Lobatto point. Numerically, this is written as

$$c_{i,j} = \frac{d}{dx} l_j(x_i^l)$$

where the indexes i and j run through the solution points in a one-dimensional line. Note that operations in two-dimensions are completed in a one-dimensional manner. Derivatives are taken of the polynomials (like those in Table A.1) and evaluated at the Gauss-Lobatto points. The correction coefficients are derived from Equation (2.1.13), which was written as

$$\int_{V_m} w \boldsymbol{\delta}_m dV = \int_{\partial V_m} w \left[\boldsymbol{F}_{com}^n - \boldsymbol{F}^n(\boldsymbol{q}_m) \right] dS$$

The correction polynomial is just a linear combination of correction coefficients and Lagrange polynomials, written as

$$\boldsymbol{\delta}_m = \sum_{i=1}^{k+1} \boldsymbol{\alpha}_i l_i(x)$$

Table A.1: Lagrange polynomials for CPR/NDG P^2 reconstruction

j	1	2	3
$l_j(x)$	$\frac{x(x-1)}{2}$	$-(x-1)(x+1)$	$\frac{x(x+1)}{2}$

Note that the correction coefficients (α) are identical for each differential equation, while the polynomial is a function of the normal flux difference. Depending on how the correction polynomial is selected, different correction coefficients can be obtained. In this thesis, the Radau polynomials are considered to due the higher accuracy compared to other polynomials [17]. The formulation for solving the correction coefficients for P^2 reconstruction becomes the following:

$$\int_{V_m} w\boldsymbol{\delta}_m dV = \begin{bmatrix} \int_{-1}^{1} l_1 l_1 & \int_{-1}^{1} l_1 l_2 & \int_{-1}^{1} l_1 l_3 \\ \int_{-1}^{1} l_2 l_1 & \int_{-1}^{1} l_2 l_2 & \int_{-1}^{1} l_2 l_3 \\ \int_{-1}^{1} l_3 l_1 & \int_{-1}^{1} l_3 l_2 & \int_{-1}^{1} l_3 l_3 \end{bmatrix} \begin{pmatrix} \alpha_{L,1} \\ \alpha_{L,2} \\ \alpha_{L,3} \end{pmatrix} = \begin{pmatrix} l_1(-1) \\ l_2(-1) \\ l_3(-1) \end{pmatrix}$$

Inserting numeric values, the equation becomes the following:

$$\int_{V_m} w\boldsymbol{\delta}_m dV = \begin{bmatrix} 0.2667 & 0.1333 & -0.0667 \\ 0.1333 & 1.0667 & 0.1333 \\ -0.0667 & 0.1333 & 0.2667 \end{bmatrix} \begin{pmatrix} \alpha_{L,1} \\ \alpha_{L,2} \\ \alpha_{L,3} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Due to symmetry, $\alpha_{L,j} = \alpha_{R,k+2-j}$, and only one set of coefficients need to be derived. Note that this formulation is identical to the Nodal Discontinuous Galerkin (NDG) approach for the lifting matrix in Equation (2.1.34).

DG Coefficients

In the Discontinuous Galerkin (DG) method, three sets of coefficients are needed for each order of accuracy: Interpolation to edges, stiffness matrix, and a mass matrix. The basis is again formed by Lagrange polynomials written as

$$l_j(x) = \prod_{s=1, s\neq j}^{k+1} \left(\frac{x - x_s^g}{x_j^g - x_s^g} \right)$$

where x^g are Gauss-Legendre points within an element. In order to interpolate the solution to the edges, a simple substitution is needed in the above formulation

$$c_{L,j} = l_j(-1) \tag{A.0.1}$$

where c_L is the interpolation coefficients to the left interface. The interpolation to the right interface follows a substitution of $x = 1$. Example computations for P^2 reconstruction are shown in Table A.2. The mass and stiffness matrices require integrals and derivatives of the Lagrange polynomials. The mass matrix, for P^2 reconstruction is written as

$$[M_{i,j}] = \begin{bmatrix} \int_{-1}^{1} l_1 l_1 & \int_{-1}^{1} l_1 l_2 & \int_{-1}^{1} l_1 l_3 \\ \int_{-1}^{1} l_2 l_1 & \int_{-1}^{1} l_2 l_2 & \int_{-1}^{1} l_2 l_3 \\ \int_{-1}^{1} l_3 l_1 & \int_{-1}^{1} l_3 l_2 & \int_{-1}^{1} l_3 l_3 \end{bmatrix}$$

while the stiffness matrix is

$$[S_{i,j}] = \begin{bmatrix} \int_{-1}^{1} l_1 \frac{\partial l_1}{\partial x} & \int_{-1}^{1} l_1 \frac{\partial l_2}{\partial x} & \int_{-1}^{1} l_1 \frac{\partial l_3}{\partial x} \\ \int_{-1}^{1} l_2 \frac{\partial l_1}{\partial x} & \int_{-1}^{1} l_2 \frac{\partial l_2}{\partial x} & \int_{-1}^{1} l_2 \frac{\partial l_3}{\partial x} \\ \int_{-1}^{1} l_3 \frac{\partial l_1}{\partial x} & \int_{-1}^{1} l_3 \frac{\partial l_2}{\partial x} & \int_{-1}^{1} l_3 \frac{\partial l_3}{\partial x} \end{bmatrix}$$

For P^2 reconstruction, these matrices become the following:

$$[M_{i,j}] = \begin{bmatrix} 0.556 & 0.0 & 0.0 \\ 0.0 & 0.8889 & 0.0 \\ 0.0 & 0.0 & 0.5556 \end{bmatrix}$$

$$[S_{i,j}] = \begin{bmatrix} -1.0758 & 1.4344 & -0.3586 \\ -0.5738 & 0.0 & 0.5738 \\ 0.3586 & -1.4344 & 1.0758 \end{bmatrix}$$

These matrices are only for one-dimensional elements. For two-dimensional elements, additional dimensions are needed to integrate points in the second dimension. This generates matrices that are size $(k+1)^2 \times (k+1)^2$ for a P^k reconstruction.

Table A.2: Left interface DG interpolation coefficients

j	1	2	3
$c_{L,j}$	1.4788	-0.6667	0.1878

SD Coefficients

For the Spectral Difference (SD) method, two sets of coefficients are required, interpolation and derivative coefficients. However, two sets of Lagrange polynomials need to be defined. They are written as

$$l_j^l(x) = \prod_{s=1, s\neq j}^{k+2} \left(\frac{x - x_s^l}{x_j^l - x_s^l} \right)$$

$$l_j^g(x) = \prod_{s=1, s\neq j}^{k+1} \left(\frac{x - x_s^g}{x_j^g - x_s^g} \right)$$

where l^l are the Lagrange polynomials for the flux points (Gauss-Lobatto points) and l^g are the Lagrange polynomials for the solution points (Gauss-Legendre points). For a P^2 reconstruction, the interpolation coefficients become the following:

$$c_{i,j} = \begin{bmatrix} l_1^g(x_1^l) & l_1^g(x_2^l) & l_1^g(x_3^l) & l_1^g(x_4^l) \\ l_2^g(x_1^l) & l_2^g(x_2^l) & l_2^g(x_3^l) & l_2^g(x_4^l) \\ l_3^g(x_1^l) & l_3^g(x_2^l) & l_3^g(x_3^l) & l_3^g(x_4^l) \end{bmatrix}$$

where the Lagrange polynomials at the solution points are evaluated at the flux point locations. The derivative coefficients can be computed in the following manner:

$$d_{i,j} = \begin{bmatrix} \frac{\partial}{\partial x} l_1^l|_{x_1^g} & \frac{\partial}{\partial x} l_1^l|_{x_2^g} & \frac{\partial}{\partial x} l_1^l|_{x_3^g} \\ \frac{\partial}{\partial x} l_2^l|_{x_1^g} & \frac{\partial}{\partial x} l_2^l|_{x_2^g} & \frac{\partial}{\partial x} l_2^l|_{x_3^g} \\ \frac{\partial}{\partial x} l_3^l|_{x_1^g} & \frac{\partial}{\partial x} l_3^l|_{x_2^g} & \frac{\partial}{\partial x} l_3^l|_{x_3^g} \\ \frac{\partial}{\partial x} l_4^l|_{x_1^g} & \frac{\partial}{\partial x} l_4^l|_{x_2^g} & \frac{\partial}{\partial x} l_4^l|_{x_3^g} \end{bmatrix}$$

where derivatives are performed on the Lagrange polynomials at the flux points and evaluated at the solution points.

BIBLIOGRAPHY

[1] Wang Z.J., Fidkowski K., Abgrall R., Bassi F., Caraeni D., Cary A., Deconinck H., Hartmann R., Hillewaert K., Huynh H.T., Kroll N., May G., Persson P.O., Van Leer B., and Visbal M. High-order CFD methods: current status and perspective. *International Journal for Numerical Methods in Fluids*, 72:811–845, 2013.

[2] Wang Z.J. and Gao H. Spectral (finite) volume method for conservation laws on unstructured grids: basic formulation. *Journal of Computational Physics*, 178:210–251, 2002.

[3] NVIDIA. *NVIDIA CUDA C Programming Guide*, volume 7.0. 2015.

[4] Castro M., Ortega S., de la Asunción M., Mantas J.M., and Gallardo J.M. GPU computing for shallow water flow simulation based on finite volume schemes. *High Performance Computing*, 339:165–184, 2011.

[5] Obenschain K., Corrigan K., and Patnaik G. Performance of unstructured finite volume code on a cluster with multiple GPUs per node. *AIAA*, (2011-945), 2011.

[6] Bassi F. and Rebay S. High-order accurate discontinuous finite element solution of the 2d Euler equations. *Journal of Computational Physics*, 138:251–285, 1997.

[7] Bassi F. and Rebay S. A high-order accurate discontinous finite element method for the numerical solution of compressible Navier-Stokes equations. *Journal of Computational Physics*, 131:267–279, 1997.

[8] Baumann C.E. and Oden T.J. A discontinuous hp finite element method for the euler and Navier-Stokes equations. *International Journal for Numerical Methods in Fluids*, 31:79–95, 1999.

[9] Cockburn B. and Shu C.W. The Runge-Kutta discontinuous Galerkin method for conservation laws V: multidimensional systems. *Journal of Computational Physics*, 141:199–224, 1998.

[10] Cockburn B. and Shu C.W. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws II: general framework. *Mathematics of Computation*, 52:411–435, 1989.

[11] Cockburn B., Lin S., and Shu C.W. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws iii: one-dimensional systems. *Journal of Computational Physics*, 84:90–113, 1989.

[12] Reed W. and Hill T. Triangular mesh methods for the neutron transport equation. Report, Los Alamos Scientific Laboratory, 1973.

[13] Van Leer B. and Nomura S. Discontinuous Galerkin for diffusion. *AIAA*, (2005-5108), 2005.

[14] Nastase C.R. and Mavriplis D.J. High-order discontinous Galerkin methods using an hp-multigrid approach. *Journal of Computational Physics*, 213:330–357, 2006.

[15] Hesthaven J.S. and Warburton T. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications.* Springer-Verlag, New York, 2008.

[16] Klöckner A., Warburton T., Bridge J., and Hesthaven J.S. Nodal discontinuous Galerkin methods on graphics processors. *Journal of Computational Physics*, 228:7863–7882, 2009.

[17] Huynh H.T. A flux reconstruction approach to high-order schemes including discontinuous Galerkin methods. *AIAA*, (2007-4079), 2007.

[18] Wang Z.J. *Adaptive High-Order Methods in Computational Fluid Dynamics*. 2011.

[19] Wang Z.J. and Gao H. A unifying lifting collocation penalty formulation including the discontinuous Galerkin, spectral volume/difference methods for conservation laws on mixed grids. *Journal of Computational Physics*, 228:8161–8186, 2009.

[20] Yu M.L. and Wang Z.J. On the connection between the correction and weighting functions in the correction procedure via reconstruction method. *Journal of Scientific Computing*, 54 (1):227–244, 2013.

[21] Hoffmann M., Munz C-.D., and Wang Z.J. Efficient implementation of the CPR formulation for the Navier-Stokes equations on GPUs. *ICCFD*, (7-2603), 2012.

[22] Zimmerman B.J. and Wang Z.J. The efficient implementation of correction procedure via reconstruction with graphics processing unit computing. *Computers and Fluids*, 101:263–272, 2014.

[23] Liu Y., Vinokur M., and Wang Z.J. Discontinuous spectral difference method for conservation laws on unstructured grids. *Journal of Computational Physics*, 216:780–801, 2006.

[24] May G. and Jameson A. A spectral difference method for the Euler and Navier-Stokes equations. *AIAA*, (2006-304), 2006.

[25] Sun Y. and Wang Z.J. High-order multidomain spectral difference method for the Navier-Stokes equations on unstructured hexahedral grids. *Journal of Computational Physics*, 2:301–333, 2007.

[26] Zimmerman B.J., Wang Z.J., and Visbal M. High-order spectral difference: verification and acceleration using GPU computing. *AIAA*, (2013-2941), 2013.

[27] Yu M.L., Wang Z.J., and Liu Y. On the accuracy and efficiency of discontinuous Galerkin, spectral difference and correction procedure via reconstruction methods. *Journal of Computational Physics*, 259:75–95, 2014.

[28] Zimmerman B. J., Regele J. D., and B. Wie. A comparative study of 2D numerical methods with GPU computing. Manuscript to be submitted, 2016.

[29] Cockburn B. and Shu C.W. Runge-Kutta discontinuous Galerkin methods for convection dominated problems. *Journal of Scientific Computing*, 16:173–261, 2001.

[30] Kubatko E., Dawson C., and Westerink J. Time step restrictions for Runge-Kutta discontinuous Galerkin methods on triangular grid. *Journal of Computational Physics*, 227:9697–9710, 2008.

[31] Richtmyer R. D. Neumann, V. A method for the numerical calculation of hydrodynamic shocks. *Journal of Applied Physics*, 21:232–237, 1950.

[32] Burbeau A., Sagaut P., and Bruneau C-H. A problem-independent limiter for high-order Runge-Kutta discontinuous Galerkin methods. *Journal of Computational Physics*, 169:111–150, 2001.

[33] Park J.S., Yu M., Kim C., and Wang Z.J. Comparative study of shock-capturing methods for high-order CPR: MLP and artificial viscoisity. *ICCFD*, (2014-0067), 2014.

[34] Hartmann R. and Houston P. Adaptive discontinuous Galerkin finite element methods for the compressible Euler equations. *Journal of Computational Physics*, 183:508–532, 2002.

[35] Persson P.-O. and Peraire J. Sub-cell shock capturing for discontinuous Galerkin methods. *AIAA*, (2006-0112), 2006.

[36] Kawai S. and Lele S. Localized artificial diffusivity scheme for discontinuity capturing on curvalinear meshes. *Journal of Computational Physics*, 227:9498–9526, 2008.

[37] Van Leer B. Towards the ultimate conservative difference scheme II. Monotonicity and conservation combined in a second order scheme. *Journal of Computational Physics*, 14:361–370, 1974.

[38] Van Leer B. Towards the ultimate conservation difference scheme V. A second order sequel to Godunov's method. *Journal of Computational Physics*, 32:101–136, 1979.

[39] Harten A., Enquist B., Osher S., and Chagravarthy S.R. Uniformly high order accurate essentially non-oscillatory schemes, iii. *Journal of Computational Physics*, 71:231–303, 1987.

[40] Burbeau A., Sagaut P., and Bruneau Ch.-H. A problem-independent limiter for high-order Runge-Kutta discontinuous Galerkin methods. *Journal of Computational Physics*, 169:111–150, 2001.

[41] Biswas R., Devine K.D., and Flaherty J.E. Parallel, adaptive finite element methods for conservation laws. *Applied Numerical Mathematics*, 14, 1994.

[42] National Research Council. Defending planet Earth: Near-Earth object surveys and hazard mitigation strategies. Report, USNRC, 2010.

[43] Boslough M. Airburst warning and response. *IAA PDC*, (2166721), 2011.

[44] NASA Jet Propulsion Laboratory. Near Earth Object Discovery Statistics. http://neo.jpl.nasa.gov/stats. Accessed: 2016-02-08.

[45] R. B. Adams, Alexander R., Bonometti J., Chapman J., Fincher S., Hopkins R., Kalkstein M., Polsgrove T., Statham G., and White S. Survey of technologies relevant to defense from near-earth objects. Report NASA-TP-2004-213089, NASA MSFC.

[46] Holsapple K. A. About deflecting asteroids and comets. In M. et al Belton, editor, *Hazards Due to Comets and Asteroids*, pages 113–140. Cambridge University Press, 2005.

[47] Ahrens T. J. and Harris A. W. Deflection and fragmentation of near-earth asteroids. In T Gehrels, editor, *Hazards Due to Comets and Asteroids*, pages 897–927. The University of Arizona Press, Tuscon, AZ, 1994.

[48] Sanchez J., Vasile M., and Radice G. On the consequences of a fragmentation due to a NEO mitigation strategy. *59th International Astronautical Congress*, (IAC-08-C1.3.10), 2008.

[49] Kaplinger B., Wie B., and Dearborn D. Nuclear fragmentation/dispersion modeling and simulation of hazardous near-Earth objects. *Acta Astronautica*, 90:156–164, 2013.

[50] Premaratne P. Nuclear subsurface explosion modeling and hydrodynamic fragmentation simulation of hazardous asteroids. Master's thesis, Iowa State University, 2014.

[51] Hérault A., Bilotta G., and Dalrymple R. A. SPH on GPU with CUDA. *Journal of Hydraulic Research*, 48:74–79, 2010.

[52] Agertz O. *et al.* Fundamental differences between SPH and grid methods. *Monthly Notices of the Royal Astronomical Society*, 380(3):963–978, 2007.

[53] Abadi M. G., Moore B., and Bower R. G. Ram pressure stripping of spiral galaxies in clusters. *Monthly Notices of the Royal Astronomical Society*, 308(4):947–954, 1999.

[54] Zimmerman B. J. and B. Wie. GPU-accelerated computational tool for studying the effectiveness of asteroid disruption techniques. Manuscript submitted for publication in *Acta Astronautica*, 2016.

[55] Zimmerman B. J. and B. Wie. A GPU-accelerated multiphase computational tool for asteroid fragmentation/pulverization simulation. Manuscript submitted for publication in *AIAA*, 2016.

[56] Zimmerman B. J. and Wie B. Computational validation of nuclear explosion energy coupling models for asteroid fragmentation. *AIAA*, (2014-4146), 2014.

[57] Zimmerman B. J. and Wie B. A GPU-accelerated computational tool for asteroid disruption modeling and simulation. *AAS*, (15-568), 2015.

[58] Zimmerman B. J. and Wie B. A GPU-accelerated multiphase computational tool for asteroid fragmentation/pulverization simulation. *AAS*, (16-242), 2016.

[59] Zimmerman B. J. and Wie B. Computational validation of nuclear explosion energy coupling models for asteroid fragmentation. *AIAA*, (2014-4146), 2014.

[60] Zimmerman B. J. and Wie B. GPU-accelerated computational tool development for studying the effectiveness of nuclear subsurface explosions. *IAA PDC*, (15-03-15), 2015.

[61] Wie B., Zimmerman B. J., and Premaratne P. A non-nuclear MKIV (multiple kinetic-energy impactor vehicle) system for dispersive pulverization of small asteroids with short warning times. *AAS*, (15-567), 2015.

[62] Lyzhoft J. and Wie B. Hypervelocity terminal guidance of a multiple kinetic-energy impactor vehicle (MKIV). *AAS*, (16-411), 2016.

[63] Wood L., Hyde R., Ishikawa M., and Teller E. Cosmic bombardment v: Threat object-dispersing approaches to active planetary defense. Report, Lawrence Livermore National Laboratory, Livermore, CA, 1995. Proceedings of the Planetary Defense Workshop.

[64] Kaplinger B., Wie B., and Dearborn D. Earth-impact modeling and analysis of a near-Earth object fragmented and dispersed by nuclear subsurface explosions. *Journal of the Astronautical Sciences*, 59:103–121, 2014.

[65] D. S. Dearborn. 21st century steam for asteroid mitigation. *AIAA*, (2004-1413), 2004.

[66] Barrera M. J. Conceptual design of an asteroid interception for a nuclear deflection mission. *AIAA*, (2004-1481), 2004.

[67] Wie B. Hypervelocity nuclear interceptors for asteroid disruption. *Acta Astronautica*, 90:146–155, 2013.

[68] Pitz A., Kaplinger B., Vardaxis G., Winkler T., and Wie B. Conceptual design of a hypervelocity asteroid intercept vehicle (HAIV) and its flight validation mission. *Acta Astronautica*, 94:42–56, 2014.

[69] Barbee B., Wie B., Steiner M., and Getzandanner K. Conceptual design of a flight demonstration mission for a hypervelocity asteroid intercept vehicle. *Acta Astronautica*, 106:139–159, 2015.

[70] Kaplinger B., Premaratne P. D., Setzer C., and Wie B. GPU-accelerated 3D modeling and simulation of a blended kinetic impact and nuclear subsurface explosion. *AIAA*, (2013-4548), 2013.

[71] Weaver R. P., Barbee B. W., Wie B., and Zimmerman B. J. Lost Alamos RAGE simulations of the HAIV mission concept. *IAA PDC*, (15-03-14), 2015.

[72] Air Force Institute of Technology. *Critical Technologies for National Defense*. 1991.

[73] Kinslow R. and Cable A. J. *High-velocity Impact Phenomena*. 1970.

[74] Prater R. F. *Hypervelocity impact - material strength effects of crater formation and shock propagation in three aluminum alloys*. PhD thesis, AFML, 1970.

[75] MacCormackR. W. The effect of viscosity in hypervelocity impact cratering. *Journal of Spacecraft and Rockets*, 40, 1969.

[76] Cloutman L. D. SPH simulations of hypervelocity impact. Report, Lawrence Livermore National Laboratory, 1991.

[77] Bjork R. L. Armour Research Foundation, Illinois Institute of Technology, 1958.

[78] Bjork R. L. Analysis of the formation of meteor crater, Arizona: A preliminary report. *Journal of Geophysical Research*, 66:3379–3387, 1961.

[79] Bjork R. L. *Proceedings Tenth International Astronautical Congress*, 2:505–514, 1960.

[80] Walsh J. M. and Tillotson J. H. Hydrodynamics of hypervelocity impact. *Proceedings Sixth Hypervelocity Impact Symposium*, 2:59–104, 1963.

[81] Walsh J. M., Johnson W. E., Dienes J. K., Tillotson J. H., and Yates D. R. Summary report on the theory of hypervelocity impact. report GA-5119, General Dynamics Corporation, 1964.

[82] Walsh J. M. and Johnson W. E. On the theory of hypervelocity impact. Technical Report 1, 1965.

[83] Riney T. D. Theoretical hypervelocity impact calulations using the PICWICK code. report R64SD13, General Electric Company Report, 1964.

[84] Heyda J. F and Riney T. D. Peak axial pressures in semi-infinite media under hypervelocity impact. report R64SD87, General Electric Company Report, 1964.

[85] Eichelberger R. J. and Gehring J. W. Effects of meteoroid impacts on space vehicles. *American Rocket Society*, 32:1583–1591, 1962.

[86] Eichelberger R. J. Hypervelocity impact. In *Behavior of Materials under Dynamic Loading*, pages 155–187. ASME, 1965.

[87] Leveque R. J. *Finite-Volume Method for Hyperbolic Problems*. 2002.

[88] Needham C. E. *Blast Waves*. Springer, 2010.

[89] Karni S. Multicomponent flow calcuations by a consistent primitive algorithm. *Journal of Computational Physics*, 112:31–43, 1994.

[90] Abgrall R. How to prevent pressure oscillations in multicomponent flow calculations: A quasi conservative approach. *Journal of Computational Physics*, 125:150–160, 1996.

[91] Abgrall R. Generalisation of the Roe scheme for the computation of mixture of perfect gases. *Recherche Aérospatiale*, 6, 1988.

[92] Larrouturou B. How to preserve the mass fraction positivity when computing compressible multi-component flows. *Journal of Computational Physics*, 95, 1991.

[93] Larrouturou B. and Fezoui L. On the equations of multi-component perfect or real gas inviscid flow. In Hanouzet Carasso, Charrier and Joly, editors, *Non-linear Hyperbolic Problems*, volume 1402, pages 69–98. Lecture Notes in Mathematics, 1989.

[94] Engquist B. and Sjogreen B. Robust difference approximations of stiff inviscid detonation waves. 1991.

[95] Harabetian E. and R. Pego. Efficient hybrid shock capturing schemes. *IMA Preprints Series*, 1990.

[96] Hou T. Y. and LeFloch P.G. Why nonconservative schemes converge to wrong solutions. Error analysis. *Mathematics of Computation*, 62:497–530, 1994.

[97] Allaire G., Clerc S., and Kokh S. A five-equation model for the simulation of interfaces between compressible fluids. *Journal of Computational Physics*, 181:577–616, 2002.

[98] Glimm J., Grove J., Li X., and Tan D. Robust computational algorithms for dynamic interface tracking in three dimensions. *SIAM Journal of Scientific Computing*, 21, 2000.

[99] Unverdi S. O. and Tryggvason G. A front tracking method for viscous, incompressible, multifluid flows. *Journal of Computational Physics*, 100, 1992.

[100] Hirt C. W. and Nichols B. D. Volume of fluid (VOF) method for the dynamics of free boundaries. *Journal of Computational Physics*, 39, 1981.

[101] Lafaurie B., Nardone C., Scardovelli R., Zaleski S., and Zanetti G. Modelling merging and fragmentation in multiphase flows with SURFER. *Journal of Computational Physics*, 113, 1994.

[102] Scardovelli R. and Zaleski S. Direct numerical simulation of free-surface and interfacial flow. *Annual Review of Fluid Mechanics*, 31, 1999.

[103] Osher S. and Sethian J. Front propagating with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 78, 1988.

[104] Gryngarten L. D. and Menon S. Shock-bubble interaction simulations using a new two-phase discontinuous Galerkin method. *AIAA*, (2011-294), 2011.

[105] L. Sainsaulieu. Finite volume approximations of two phase-fluid flows based on an approximate Roe-type Riemann solver. *Journal of Computational Physics*, 121, 1995.

[106] Stewart H. and Wendroff B. Two-phase flow: Models and methods. *Journal of Computational Physics*, 56, 1984.

[107] Saurel R. and Abgrall R. A multiphase Godunov method for compressible multifluid and multiphase flows. *Journal of Computational Physics*, 150, 1999.

[108] Abgrall R. Généralisation du schéma de roe pour le calcul d'écoulements de mélanges de gaz à concentrations variables. *Recherche Aérospatiale*, 6, 1998.

[109] Saurel R. and Abgrall R. A simple method for compressible multifluid flows. *SIAM Journal of Scientific and Statistical Computing*, 32:1115–1145, 1999.

[110] Shyue K-M. An efficient shock-capturing algorithm for compressible multicomponent problems. *Journal of Computational Physics*, 142:208–242, 1998.

[111] Tannehill J.C., Anderson A.A., and Pletcher R.H. *Computational Fluid Mechanics and Heat Transfer*, volume 2. 1997.

[112] Roe P.L. Characteristic-based schemes for the Euler equations. *Annual Review of Fluid Mechanics*, 18:337–365, 1986.

[113] Zienkiewicz O.C. and Tayler R.C. *The Finite Element Method the Basics*, volume 1. 2000.

[114] Rusanov V.V. Calculation of interaction of non-steady shock waves with obstacles. *Journal of Computational Mathematics and Mathematical Physics*, USSR(1):267–279, 1961.

[115] Harten A., Lax P. D., and van Leer B. On upstream differencing and Godunov-type schemes for hyperbolic conservation laws. *SIAM Journal on Numerical Analysis*, 24(2):279–309, 1987.

[116] Spruce M. Toro E. F. and Speares W. Restoration of the contact surface in the HLL-Riemann solver. Report, Department of Aerospace Science, Collegue of Aeronautics, Cranfield Institude of Technology, UK, 1992.

[117] Spruce M. Toro E. F. and Speares W. Restoration of the contact surface in the HLL-Riemann solver. *Shock Waves*, 4:25–34, 1994.

[118] Toro E.F. *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*. Springer, London/New York, 2009.

[119] Davis S. F. Simplified second-order Godunov-type methods. *SIAM Journal of Scientific Computing*, 9:445–473, 1988.

[120] Shu C.W. Total-variation-diminishing time discretizations. *SIAM Journal of Scientific Computing*, 9:1073–1084, 1988.

[121] Taubench. http://www.ipacs-benchmark.org/index.php?s=download&unterseite=taubench.

[122] Shu C.W. *Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws, in: Advanced Numerical Approximation of Nonlinear Hyperbolic Equations*. Springer-Verlag, Berlin/New York, 1998.

[123] Wang Z.J., Liu Y., May G., and A. Jameson. Spectral difference method for unstructured grids II: Extension to the Euler equations. *Journal of Scientific Computing*, 32(1), 2006.

[124] Haga T., Gao H., and Wang Z.J. A high-order unifying discontinuous formulation for 3-*d* mixed grids. *AIAA*, (2010-540), 2010.

[125] Wang Z. J. and Liu Y. The spectral different method for the 2*d* Euler equations on unstructered grids. *AIAA*, (2005-5112), 2005.

[126] Gubounov S. K., Zabrodine A., Ivanov M., A. Kraiko, and Prokopov G. *Résolution numérique des problèmes multidimensionnels de la dynamique des gaz*. Editions Mir, Moscow, 1979.

[127] Harlow F. and Amsden A. Fluid Dynamics. report Monograph LA-4700, Los Alamos National Laboratory, 1971.

[128] Shyue K-M. A fluid-mixture type algorithm for compressible multicomponent flow with Mie-Grüneisen equation of state. *Journal of Computational Physics*, 171:678–707, 2001.

[129] Stowe R. L. Strength and deformation properties of granite, basalt, limestone, and tuff at various loading rates. Report, Corps of Engineers, 1969.

[130] Glassrone S. and Dolan P.J. The Effects of Nuclear Weapons. report, United State Department of Defense and Energy Research and Development Administration, 1977.

[131] Sod G. A. A survey of several finite difference methods of nonlinear hyperbolic conservation laws. *Journal of Computational Physics*, 27:1–31, 1978.

[132] Henry de Frahan M. T. and Johnsen E. Discontinuous Galerkin method for multifluid Euler equations. *AIAA*, (2013-2595), 2013.

[133] Johnsen E. and Colonius T. Implementation of WENO schemes in compressible multicomponent flow problems. *Journal of Computational Physics*, 219:715–732, 2006.

[134] Rider W. J. An adaptive Riemann solver using a two-shock approximation. *Computers and Fluids*, 28:741–777, 1999.

[135] National Academy of Sciences. Effects of nuclear earth-penetrator and other weapons. Report, 2005.

[136] Gittings M. *et al.* The RAGE radiation-hydrodynamic code. *Computational Science and Discovery*, 1, 2008.

[137] Saurel R., Le Metayer O., Massoni J., and Gavrilyuk S. Shock jump relations for multiphase mixtures with stiff mechanical relaxation. *Shock Waves*, 16:209–232, 2007.

[138] Weaver R. P. personal communication, March 2015.